Escuela
Politécnica
Superior

Control dinámico multiarticular con optimización aplicado al robot TIAGo

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Álvaro Belmonte Baeza
Tutores:
Jorge Pomares Baeza
Jordi Pagès Marco

Julio 2019

Universitat d'Alacant
Universidad de Alicante

Control dinámico multiarticular con optimización aplicado al robot TIAGo

Autor

Álvaro Belmonte Baeza

Tutores

Jorge Pomares Baeza

Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal

Jordi Pagès Marco

Jefe de producto de TIAGo en PAL Robotics

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

Alicante, Julio 2019

Resumen

Este proyecto se ha realizado en colaboración con la empresa PAL Robotics para la optimización de los controladores empleados en el robot TIAGo. Este robot dispone de un manipulador de 7 grados de libertad, así como un sistema de visión para captar información de su espacio de trabajo, entre otros componentes.

En primer lugar, se ha llevado a cabo un estudio del estado del arte actual en lo que a control de robots se refiere, haciendo especial hincapié en el control dinámico de robots y sus distintas vertientes, como son el control dinámico articular, cartesiano, control visual y control óptimo.

Tras esto, se ha realizado el desarrollo teórico y análisis de los controladores dinámicos multiarticulares implementados, con el fin de justificar su empleo y validarlos matemáticamente, observando así sus características principales, así como los parámetros clave para su buen desempeño, lo cual permitirá dar una explicación razonada a los resultados obtenidos en la fase de experimentación.

Una vez terminado el desarrollo teórico, se ha llevado a cabo la simulación del robot TIAGo. En concreto, se ha realizado una simulación cinemática y dinámica del robot, con el fin de analizar el funcionamiento de sus controladores por defecto y estudiar las situaciones en las que un controlador óptimo podría mejorar su desempeño. Con este fin, se han usado el simulador Gazebo y el paquete de software ROS Control para realizar la implementación de los controladores estudiados y poder implantarlos en el robot.

A continuación, se ha abordado la implementación y simulación de controladores dinámi-

cos multiarticulares tales como los PD con prealimentación, PD+, control Par-Calculado, así como controladores cartesianos. Para cada uno de ellos, se ha realizado el ajuste correspondiente y se han contrastado las situaciones en las que mejora el desempeño del robot respecto a los controladores originales. Además, se han diseñado controladores óptimos que tratan de mejorar tanto el comportamiento del manipulador como la energía y esfuerzo necesarios para el seguimiento de trayectorias. Este ahorro de energía puede ser crítico al tratarse de un robot manipulador móvil, extendiendo así su autonomía.

Finalmente, tras la simulación y evaluación de los controladores dinámicos diseñados, se ha llevado a cabo la implementación de los mismos en el robot real. El proyecto es cotutorizado por el Jefe de Producto de TIAGo de PAL robotics, lo que ha permitido desarrollar, implementar y evaluar "in situ" los controladores diseñados durante el proyecto, contrastando y corroborando de este modo las mejoras conseguidas en el robot real.

Preámbulo: Motivación, justificación y objetivo general

La elección de este tema para el desarrollo de mi Trabajo Fin de Grado vino motivada en primera instancia por el gran interés que me despertó el campo del control de robots durante el tercer curso del grado, puesto que se trataba de uno de los pilares de la robótica al encargarse del correcto movimiento del robot, todo ello respaldado por un fuerte análisis matemático, que justifica el uso de los distintos controladores existentes.

Sin embargo, tras poner en común mis inquietudes con mi tutor y establecer una serie de objetivos para el proyecto en este campo, nos dimos cuenta de que a pesar de lo interesante del tema escogido, queríamos demostrar que más allá del componente teórico, el control dinámico de robots es de gran interés en aplicaciones prácticas.

Por ello, nos pusimos en contacto con la empresa PAL Robotics, lider mundial en robots humanoides, y concretamente con mi actual co-tutor, Jordi Pagès, para trasladarles nuestras inquietudes y comprobar el interés que despertaba esta línea de trabajo en una empresa de tal magnitud, y la respuesta no pudo ser más alentadora, dado que desde el primer momento se mostró un gran interés en el proyecto planteado.

Por este motivo, se decidió llevar a cabo la implementación de los controladores desarrollados matemáticamente en el robot TIAGo, el cual permite al usuario modificar sus controladores, con el objetivo principal de trasladar a una plataforma real el diseño teórico de controladores dinámicos multiarticulares, y más específicamente, del control óptimo de robots manipuladores.

Agradecimientos

En primer lugar, quisiera agradecer a mi tutor, Jorge, su apoyo y motivación a lo largo de todo el proyecto, respetando siempre "mis tiempos" y confiando ciegamente en mí y en mis capacidades, pasando durante este periodo de ser un profesor a ser un mentor, un referente y un amigo. Además, quisiera agradecer a Gabriel, Carlos, Andrés, y por extensión a todo el grupo de investigación *Human Robotics* por su ayuda a lo largo del desarrollo del proyecto, así como por acogerme en el grupo tanto a mí como a mis compañeros como uno más, dotando al laboratorio de un ambiente muy cordial y divertido, que hacía más amenas la largas jornadas de trabajo.

Mi más sincero agradecimiento también a mi co-tutor Jordi, así como a la empresa PAL Robotics en su conjunto, por confiar en un chaval desconocido brindándome su apoyo desinteresado siempre que fuese necesario, así como la oportunidad de trabajar por unos días con ellos en un entorno de lo más estimulante, en el cual he aprendido muchísimo tanto a nivel técnico como profesional y personal, siendo una experiencia de lo más enriquecedora.

No puedo olvidarme tampoco en estas líneas de mi familia y seres queridos, quienes han estado a mi lado tanto en lo bueno como en lo malo a lo largo de la carrera, siendo siempre ese pilar que me sostenía cuando todo lo demás fallaba. Mamá, Arantza, Javi, Paloma, Pepe, tíos, acabe donde acabe, siempre podréis contar conmigo. A ti también, Yaiza, por en tan poco tiempo ser un apoyo y una fuente de motivación y alegría constante, que me ha enseñado lo que significa realmente ser feliz. A todos, os quiero con todo mi ser.

Para terminar, he querido reservar estos últimos agradecimientos a mis compañeros de clase. Mejor dicho, a mis amigos, mis compañeros durante esta aventura, a los que siempre guardaré en mi corazón. A todos vosotros, gracias por hacerme descubrir el verdadero valor de la amistad, y por enseñarle a este cabezota insufrible que con vosotros de mi lado no hay nada que se nos resista. Espero, deseo, y haré todo lo posible para que nuestros caminos siempre se mantengan unidos, porque lo mejor que me ha dado la carrera ha sido poder compartirla con vosotros.

A tu, iaia, per criar-me en l'amor més pur que pogués somiar.
A tu, iaio, per educar-me en la disciplina que em guia cada dia.

A ti, mamá, por estar siempre a mi lado. Te debo todo lo que soy.

El presente es de ellos; el futuro,
para lo que realmente trabajé,
es mío.

Nikola Tesla.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

La robótica tiene profundas raíces culturales. A lo largo de la historia, los seres humanos hemos intentado encontrar sustitutos que sean capaces de imitar nuestro comportamiento en diversas situaciones de interacción con nuestro entorno.

Una de las grandes ambiciones del ser humano ha sido la de *dar vida* a sus artefactos. La leyenda del titán *Prometeo*, quien moldeo la raza humana con arcilla [4], así como la del gigante *Talos*, el esclavo de bronce forjado por Hefesto, demuestran como incluso la mitología griega ya estaba influida por esta ambición, la cual ha sido revisitada en multitud de ocasiones, como por ejemplo en la historia de *Frankenstein* [5] en épocas más recientes.

Así como el gigante Talos fue confiado con la tarea de proteger la isla de Creta de los invasores [6], en la *Edad Industrial* se ha delegado en un artefacto mecánico como un autómata la tarea de sustituir al ser humano en ciertas labores. Este concepto fue introducido por el dramaturgo checo Karel Čapek, quien escribió la obra *Rossum's Universal Robots* en 1920 [7]. Es en esta obra donde se acuña el término *robot*, para denominar al autómata construido por Rossum que acaba alzándose contra la humanidad en esta historia de ciencia ficción. En la figura 1.1, se puede observar el cartel publicitario de dicha obra.

Sin embargo, los robots de Rossum eran criaturas hechas de material orgánico, y no fue hasta la década de 1940 cuando el novelista ruso Isaac Asimov concibió el robot como un autómata de apariencia humana, pero falto de sentimientos, cuyo comportamiento venía dictado por la programación del ser humano, con el objetivo de cumplir ciertas normas éticas: Las *Tres leyes fundamentales de la robótica* [8].

Figura 1.1: Póster de la obra de teatro, representada con marionetas en el Federal Theatre Project, 1936-39. Fuente: Google Images.

Estas leyes establecieron reglas de comportamiento a considerar como especificaciones en el diseño de un robot, que desde entonces ha tenido la connotación de ser un producto industrial diseñado por ingenieros o técnicos especializados.

La ciencia ficción, pues, ha influenciado al hombre y mujer de la calle que continua imaginando a los robots como humanoides capaces de hablar, caminar, ver y escuchar, con una apariencia muy similar a la mostrada en las películas *Metrópolis*, precursora de la cinematografía moderna sobre robots, y más recientemente en largometrajes como *Star Wars* o *Yo, Robot*, esta última inspirada por las novelas de Asimov.

Por otro lado, si interpretamos desde un punto científico toda esta ciencia ficción, se define al robot como una máquina que, independientemente de su exterior, es capaz de interactuar y modificar el entorno en el que opera. Esto se consigue llevando a cabo tareas que vienen condicionadas por ciertas reglas de comportamiento intrínsecas a la máquina, así como por datos adquiridos por el robot sobre su estado actual y el del entorno. De hecho, el término *robótica* es habitualmente definido como ***la ciencia que estudia la conexión inteligente entre percepción y acción***.

Siguiendo con esta definición, un sistema robótico es en realidad un sistema complejo, que podemos descomponer en múltiples subsistemas, tal y como se muestra en la figura 1.2.

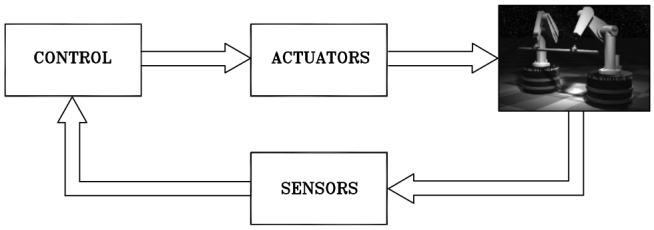


Figura 1.2: Componentes de un sistema robótico. Fuente: Siciliano, B. *Robotics: Modelling, Planning and Control*

El componente esencial de un robot es el sistema mecánico dotado, generalmente, de un aparato locomotor y/o un aparato manipulador, como ilustran los manipuladores montados sobre base móvil de la figura 1.2, una versión muy clásica con una estructura similar al robot TIAGo empleado en este proyecto.

Por su parte, la capacidad de ejercer una acción sobre el entorno, ya sea locomoción o manipulación, queda a cargo del subsistema de actuación, que consigue *animar* los componentes mecánicos del robot. El concepto de este sistema se entiende dentro del contexto del **control de movimiento**, tratando con servomotores, *drivers* y transmisiones.

En tercer lugar, la capacidad de percepción viene proporcionada por un sistema sensorial, el cual puede adquirir datos tanto del estado interno del sistema mecánico (sensores propioceptivos), como del entorno del robot (mediante el uso de cámaras o sensores de esfuerzo).

Sin embargo, el subsistema clave para la interconexión entre acción y percepción de manera inteligente es el **sistema de control**, capaz de ejecutar una determinada acción según los objetivos establecidos por una técnica de planificación de tareas, en conjunto con las restricciones impuestas al robot por parte de su entorno.

Por este motivo, la investigación en la mejora de las técnicas de control empleadas por los sistemas robóticos tiene una importancia capital, al ser determinante en el máximo aprovechamiento del resto de subsistemas que conforman un robot en su conjunto. Por muy buena que sea la mecánica del robot, lo sofisticados que sean sus sensores y actuadores, de nada

sirven por si solos sin un elemento central de control que los utilice de manera eficaz.

Así pues, con esta reseña histórica introductoria, se ha querido probar que, aun partiendo de supuestos mitológicos o basados en la ciencia ficción, todo termina por converger en la necesidad de un sistema de control lo más óptimo posible para maximizar los resultados de nuestro robot, y de este supuesto partimos para establecer, a continuación, los objetivos planteados para este proyecto.

1.1. Objetivos

El objetivo principal del proyecto desde su inicio no era otro que el de conseguir dar un paso más allá en el diseño de controladores para robots, concretamente para robots manipuladores, al ser a día de hoy los más habituales.

Así pues, en este proyecto se pretende analizar, desarrollar, comprender y justificar diversos tipos de controladores dinámicos multiarticulares para robots, demostrando que es interesante a la par que necesario el hecho de trabajar en la mejoría de los controladores de robots, más allá del PID estándar, con el fin de seguir mejorando la precisión de estas máquinas, así como sus capacidades de adaptación e interacción con el entorno.

Además, el segundo objetivo clave del proyecto es estudiar en primera persona el proceso de implantación de este tipo de controladores en un robot real de alta gama como es el TIAGo, experimentando así las distintas fases del desarrollo de cualquier tipo de software para un sistema robótico: Análisis teórico, implementación, validación en simulación y salto a la plataforma robótica real.

1.2. Estructura del proyecto

Este proyecto se divide en varios capítulos, en los cuales contaremos con distintas secciones y subsecciones con el fin de separar de forma clara los distintos conceptos tratados.

Así pues, en este primer capítulo introductorio, se ha expuesto una reseña histórica referente al origen del interés del ser humano en el campo de la robótica, así como la evolución del concepto de robot a lo largo de los años. Tras esto, se ha indicado como, basándonos en el concepto habitual de robot establecido a lo largo de los años desde la ciencia ficción, podemos darle un enfoque más científico, estableciendo así los diferentes componentes de un sistema robótico, haciendo especial hincapié en la importancia que reside en el sistema de control, cerebro de todo el conjunto en lo que a ejecución de movimiento se refiere.

A continuación, en el segundo capítulo, se hará un repaso al estado del arte actual en el campo del control de robots, desde conceptos más básicos como control cinemático, a cuestiones avanzadas como el control dinámico multiarticular, control óptimo y, en menor medida, control visual.

Tras esto, en el tercer capítulo se expondrá todo el desarrollo teórico relativo a los controladores implementados, justificando de forma matemática su validez y estabilidad. Además, también se mostrará todo lo relacionado con la implementación técnica de los controladores diseñados, tanto a nivel software como respecto a la plataforma hardware utilizada: El robot TIAGo, mostrado en la figura 1.3.

Figura 1.3: Robot TIAGo de PAL Robotics.

Una vez expuesto todo el desarrollo teórico del proyecto y su implementación, en el cuarto capítulo se analizarán los resultados obtenidos, tanto en simulación como haciendo uso del robot real, poniendo de manifiesto las diferencias entre un entorno ideal y uno real, así como la utilidad de los controladores diseñados en dicho entorno real.

Por último, en el quinto capítulo se indicarán las conclusiones finales alcanzadas tras completar el proyecto en su totalidad. Además, se hará una relación de publicaciones derivadas de la realización de este proyecto, con las que se podrá ampliar en la materia tratada a lo largo de este, para finalmente acabar con los trabajos futuros posibles partiendo de lo conseguido en el proyecto.

2. Estado del arte

En este segundo capítulo, se expondrá cual es el estado de la técnica actualmente en lo que a control de robots se refiere, con el fin de contextualizar el trabajo realizado en este proyecto, y entender mejor su encaje en el panorama investigador.

Así, en primer lugar trataremos de forma superficial conceptos básicos de control de robots, poniendo en valor su importancia en cualquier sistema robótico. Tras esto, se diferenciará entre control cinemático y control dinámico de robots, haciendo especial hincapié en esta segunda estrategia al ser la utilizada en el proyecto.

A continuación, se presentará el concepto de control óptimo, introduciendo parte de la base teórica necesaria, y mostrando algunos de los trabajos realizados en este ámbito, que como veremos se puede aplicar a muchos tipos de sistemas robóticos, más allá de los robots manipuladores, en el cual nos centraremos en el capítulo 3.

Finalmente, se comentará de forma menos detallada las diversas técnicas de control visual existentes, citando en este punto algunos de los trabajos actuales más destacados en esta técnica, que de nuevo es aplicable a multitud de campos del mundo de la robótica.

2.1. Control de robots

En cualquier aplicación de la robótica, el objetivo de completar una tarea genérica requiere de la ejecución de un movimiento específico por parte del robot. Así, la correcta ejecución de dicho movimiento es responsabilidad del sistema de control, el cual debe proveer a los actuadores del robot con las órdenes que sean necesarias para conseguir el movimiento deseado.

El control de movimiento de un robot requiere de un análisis exhaustivo de las características de la estructura mecánica, los actuadores y los sensores. El objetivo de este análisis es la obtención de modelos matemáticos que describan lo más fielmente posible la relación entrada/salida que caracteriza los distintos componentes del robot. Por tanto, el modelado del robot es una premisa necesaria para trabajar con estrategias de control de movimiento.

En lo que respecta a esta fase de modelado, se deben distinguir los dos tipos de modelo posibles: Cinemático y Dinámico. Estos conceptos se detallarán en las siguientes subsecciones.

Por tanto, el problema de controlar un robot, concretamente un robot manipulador, puede ser formulado como el de determinar el historial de fuerzas generalizadas (fuerzas o pares articulares) que deben ser realizadas por parte de los actuadores articulares (servomotores), con el fin de garantizar la ejecución de la tarea indicada, pero sin dejar de lado que se deben satisfacer ciertas condiciones tanto en régimen transitorio como en régimen permanente.

Dicha tarea puede concernir o bien la ejecución de unos movimientos específicos del robot en espacio abierto, o bien la ejecución de dichos movimientos, así como la generación de determinadas fuerzas de contacto, en el caso de que el efector final del manipulador esté restringido por el entorno. De estos dos problemas, en este trabajo tan solo se hará referencia al primero, referente al control de movimiento del robot en espacio libre. Para obtener más información en lo referente al denominado control de fuerzas, se puede acudir a textos de referencia como [9].

2.1.1. Control cinemático

El modelo cinemático de un robot describe de forma analítica la relación entre las posiciones de las distintas articulaciones que lo conforman y la posición y orientación cartesiana del efector final. En consecuencia, la *cinemática diferencial* describe la relación analítica entre el movimiento de las articulaciones del robot y la velocidad del efector final, a través de la matriz Jacobiana del manipulador. El desarrollo matemático necesario para la obtención del

modelo cinemático de un robot es algo muy extendido en la literatura, siendo los textos [9, 10] dos obras de referencia para su estudio. Un ejemplo de modelo cinemático viene ilustrado en la figura 2.1.

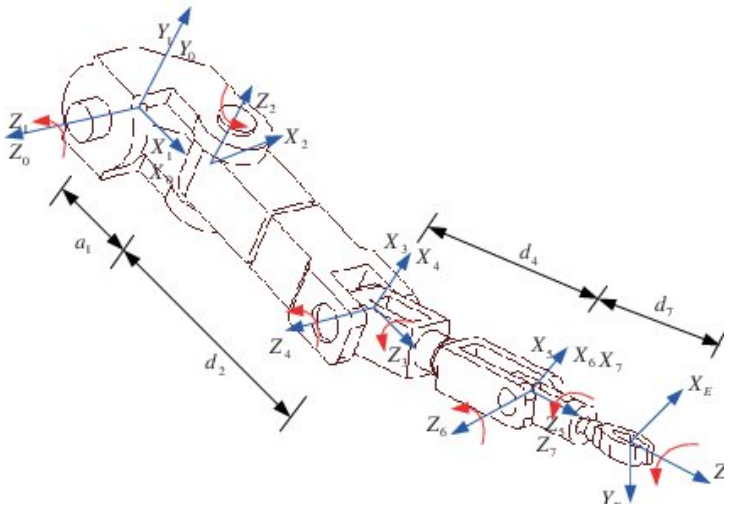


Figura 2.1: Modelo cinemático de un brazo robótico. Fuente: `www.researchgate.net`

Por su parte, el control cinemático establece cuáles son las trayectorias a seguir a lo largo del tiempo por cada una de las articulaciones, con el fin de cumplir con los objetivos fijados por parte del usuario. La obtención de estas trayectorias vendrá condicionada por las restricciones físicas propias de los accionamientos, así como de ciertos parámetros de calidad de la trayectoria, como por ejemplo suavidad o precisión.

El controlador cinemático recibe como entradas los datos procedentes del programa del robot escrito por el usuario (punto de destino, precisión, trayectoria deseada, etc.), y apoyándose en el modelo cinemático del robot, establece las trayectorias para cada articulación como variaciones de las coordenadas articulares en función del tiempo. Estas trayectorias es necesario muestrearlas con un periodo de tiempo T a decidir, generándose pues en cada instante de tiempo un vector de referencias articulares para los algoritmos de control dinámico, a más bajo nivel.

Por otro lado, este movimiento definido para el robot por parte del usuario puede ser

realizado según infinitas trayectorias espaciales. De todas ellas hay algunas que, bien por su sencillez de implementación por parte del controlador cinemático, o bien por su interés para determinadas tareas debido a características especiales de estas. Información detallada en lo relativo a la planificación de trayectorias puede encontrarse en [11].

Así pues, una vez generada la trayectoria a seguir, se necesita de un bucle de control cinemático a alto nivel que realice el seguimiento de dicha trayectoria. Para ello, se suele usar un bucle de control clásico PID, delegando en otro controlador a bajo nivel el trabajo con la dinámica del robot. En la figura 2.2 podemos observar la estructura de dicho bucle de control cinemático.

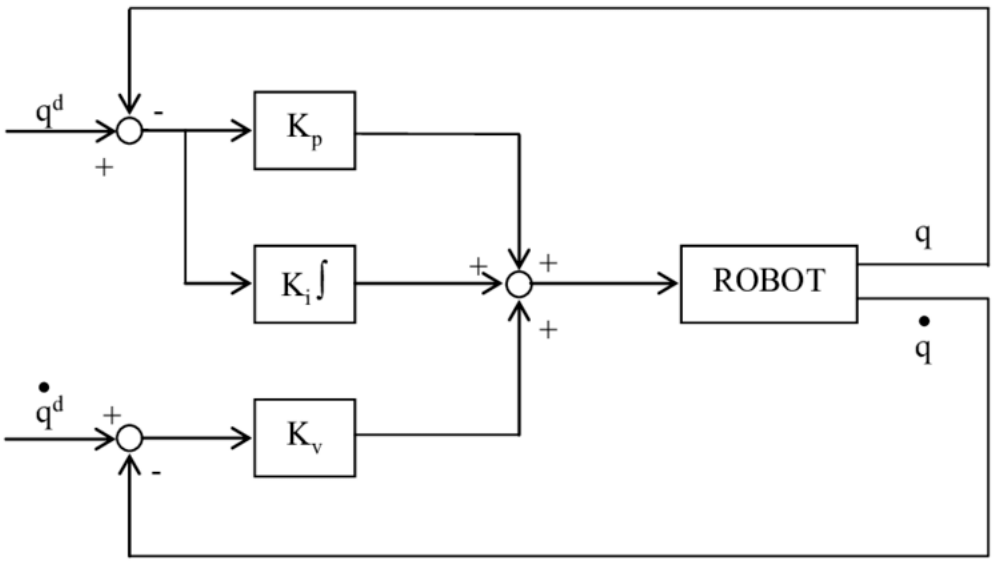


Figura 2.2: Bucle de control PID. Fuente: `www.researchgate.net`

Donde K_p es la ganancia proporcional que multiplica al error en posición de la articulación, K_v la ganancia que multiplica al error en velocidad, y finalmente K_i la ganancia que multiplica a la integral del error en posición. Por otro lado, q_d y $\dot{q_d}$ representan la posición y velocidad articular deseada respectivamente, mientras que q y $\dot{q}$ son la posición y velocidad articular actuales del robot.

Así, el uso de un controlador cinemático es adecuado para la mayoría de tareas convencionales, siempre que no contemos con velocidades o aceleraciones elevadas. Sin embargo, hay que tener en cuenta que el control cinemático selecciona las trayectorias que el robot, idealmente, debería seguir ajustándose lo más posible a las especificaciones del usuario. Sin embargo, en la práctica este ajuste del movimiento del robot no será totalmente posible debido a las características dinámicas del robot (inercias, rozamiento, transmisiones, etc.), que en la mayoría de ocasiones cuando se emplea este tipo de control son desconocidas, e impiden un seguimiento preciso entre la trayectoria deseada $\mathbf{q_d}(t)$ y la real $\mathbf{q}(t)$.

2.1.2. Control dinámico

El modelo dinámico del robot tiene como objetivo conocer la relación entre el movimiento del robot y las fuerzas aplicadas al mismo. Este modelo relaciona matemáticamente diversos factores, como son la localización del robot, ya sea en coordenadas articulares o cartesianas, con las fuerzas y pares aplicados en las articulaciones, así como con los parámetros dimensionales del robot (longitud, masas, inercias...). Dicho modelo dinámico toma, de forma genérica, la siguiente expresión:

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) \tag{2.1}$$

Donde $\boldsymbol{\tau}$ es el vector de pares articulares ejercidos en cada articulación del robot, $\boldsymbol{M}(\boldsymbol{q})$ es la matriz de inercia del robot, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ es la matriz de Coriolis que depende tanto de la posición como de la velocidad actuales del robot, y $\boldsymbol{G}(\boldsymbol{q})$ es la matriz de gravedad, que indica el efecto de esta fuerza sobre el robot en una posición determinada $\boldsymbol{q}$.

Sin embargo, la obtención del modelo dinámico no es en absoluto trivial, máxime conforme aumenta el número de grados de libertad (GDL) de la cadena cinemática, lo que lleva al empleo de métodos numéricos iterativos, siendo uno de los más conocidos el método de Newton-Euler [12]. Por tanto, el problema de la obtención del modelo dinámico de un robot es uno de los más complejos de la robótica, lo que ha provocado que sea obviado en multitud

de ocasiones, a pesar de ser imprescindible para multitud de tareas, como las recogidas en [13], y que se enumeran a continuación:

- Simulación del movimiento del robot.
- Diseño y evaluación de la estructura mecánica del robot.
- Dimensionamiento de los actuadores.
- Diseño y evaluación del control dinámico del robot.

Esta última tarea es de vital importancia, puesto que de la calidad del control dinámico del robot depende la precisión y la velocidad de sus movimientos.

El control dinámico tiene como objetivo que las trayectorias realmente seguidas por el robot $\mathbf{q}(t)$, sean lo más parecidas posibles a las propuestas por el control cinemático $\mathbf{q_d}(t)$. Para esto, se hace uso del conocimiento del modelo dinámico del robot y de las herramientas de análisis y diseño aportadas por la teoría del servocontrol (representación interna, estado, estabilidad de Lyapunov, control PID, control robusto, etc.).

Así pues, existen dos grandes grupos dentro del control dinámico articular: El control dinámico monoarticular, en el que se desprecia el acoplamiento entre los GDL del robot, y el control multiarticular, en el que se considera al robot como el sistema multivariable que en realidad es. Esta última aproximación es la que se va a tratar en este proyecto, puesto que la consideración de que cada articulación solo se ve afectada por sus propias características no es siempre aceptable, y si hacemos uso de técnicas de control acoplado que tengan en cuenta el modelo real del robot, como es el caso del control multiarticular, podremos tratar de desacoplar el sistema, como sucede en [14] con un esquema clásico observable en la figura 2.3. Estos conceptos se ampliarán en la sección 3, y pueden ser estudiados en detalle en textos de referencia como [15], o los ya mencionados [9, 10].

En el presente proyecto, se han implementado algunos de los controladores dinámicos multiarticulares más destacados, como son el control *PD con prealimentación*, *PD+* o *Par-*

Calculado. En el capítulo 3, se hará un estudio más detallado de estos controladores.

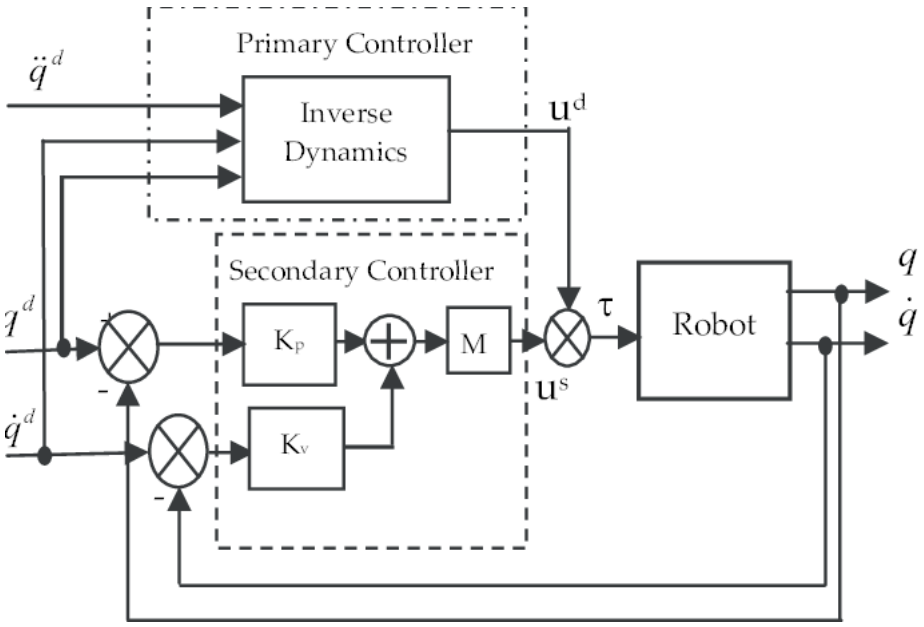


Figura 2.3: Control dinámico con desacople del sistema. Fuente: `https://www.researchgate.net/publication/272779236_SVM-Based_Control_System_for_a_Robot_Manipulator`

El control dinámico de robots manipuladores supone aún en la actualidad un campo activo de investigación dentro de los que cabe mencionar la utilización de técnicas de aprendizaje para la mejora del desempeño de los controladores dinámicos [16], uso de técnicas de resolución de redundancia para la optimización del comportamiento de robots manipuladores [17], así como el diseño de técnicas de control eficientes aplicadas a distinto tipo de robots como humanoides [18], drones [19], etc.

2.2. Control óptimo

En los últimos años, los robots manipuladores, principalmente los industriales, están formados por cadenas cinemáticas redundantes, con el objetivo de ganar versatilidad y ser capaces de llevar a cabo distintas tareas en entornos dinámicos o no estructurados.

Esta redundancia dota a los robots manipuladores con mayor número de GDL de los que

realmente necesitan para realizar la tarea, formando así complejas estructuras articulares. Gracias a esto, son capaces de sobreponerse a limitaciones intrínsecas de su construcción, y ganar mayor flexibilidad a la hora de llevar a cabo cualquier tarea. Por ejemplo, en el sector industrial, esta redundancia se ha aprovechado para satisfacer algunos requisitos propios de los procesos de manufacturación, como la manipulación de piezas de forma irregular, la necesidad de una alta precisión, o la alta velocidad a la que se someten los robots en estos entornos [20]. En la figura 2.4, se ilustra un manipulador redundante en el espacio 2D, puesto que cuenta con mayor número de GDL de los que realmente necesita para alcanzar cualquier posición en su espacio de trabajo.

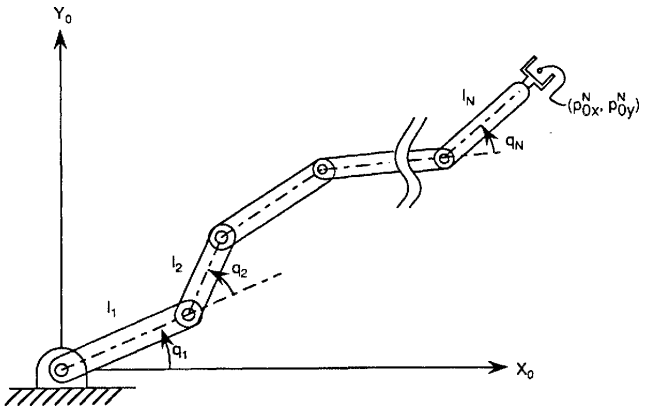


Figura 2.4: Esquema de un robot redundante en el espacio 2D. Fuente: `https://www.semanticscholar.org/paper/Input-relegation-control-for-gross-motion-of-a-Unseren/53b7ceadc367828bbdcd73361fbc60233e8fac5d`

Sin embargo, el control de un sistema redundante no involucra tan solo un movimiento adecuado del efector final, sino también conseguir movimientos coordinados de las articulaciones del robot que favorezcan la estabilidad de toda su estructura. Este es uno de los casos donde aparece el concepto de **control óptimo**, concretamente aplicado a los robots manipuladores.

En este caso, el control óptimo trata de controlar robots redundantes considerando en el proceso la dinámica del robot. De este modo, esta técnica considera la optimización de las señales motoras o pares articulares enviados al sistema mecánico durante la ejecución de la

tarea.

El control de robots redundantes, del que se deriva el control óptimo, ha sido un tema de interés recientemente, y varios autores destacados han propuesto diferentes algoritmos de control basados en dinámica. De todos ellos, cabe mencionar el trabajo presentado por Khatib en [21], donde se propone un esquema de control dinámico para robots redundantes basado en el uso de la pseudo-inversa de la matriz Jacobiana. Otros trabajos como los realizados por Xian [22] y Zergeroglu [23] se fundamentan en un controlador no lineal que asegura un seguimiento asintótico de trayectorias en el espacio cartesiano.

En [24] se propone una interpretación de la cinemática y dinámica de un robot en términos de un problema de control de seguimiento. Para ello, se basa en la aplicación del principio de Gauss para derivar la ley de control [25, 26]. Esta definición permite la generalización del problema de control como un controlador óptimo sobre el que aplicar restricciones o funciones objetivos a optimizar durante la tarea de seguimiento.

Otros trabajos más recientes se centran en el uso de Redes Neuronales debido a su capacidad de aprendizaje y de computación paralela. En este aspecto, la mayoría de las mejoras radican en la compensación de no-linearidades [27] y las incertidumbres del sistema [28]. De forma similar, en [29] se muestra cómo las redes neuronales pueden lidiar con las no-linearidades sin una fase previa de aprendizaje offline. El algoritmo de aprendizaje adaptativo propuesto se deriva del análisis de estabilidad por Lyapunov, por lo que tanto la estabilidad en el seguimiento de la trayectoria como la convergencia del error en bucle se pueden garantizar. En la figura 2.5, se puede observar el esquema de control propuesto en [29].

Además, otra de las ventajas del control óptimo es que no está restringido al control de robots manipuladores en espacio cartesiano, sino que también se puede ampliar a otros campos, como es el caso de su aplicación a robots esféricos [30], o a un péndulo invertido con ruedas [31]. Ambos casos se pueden observar en la figura 2.6.

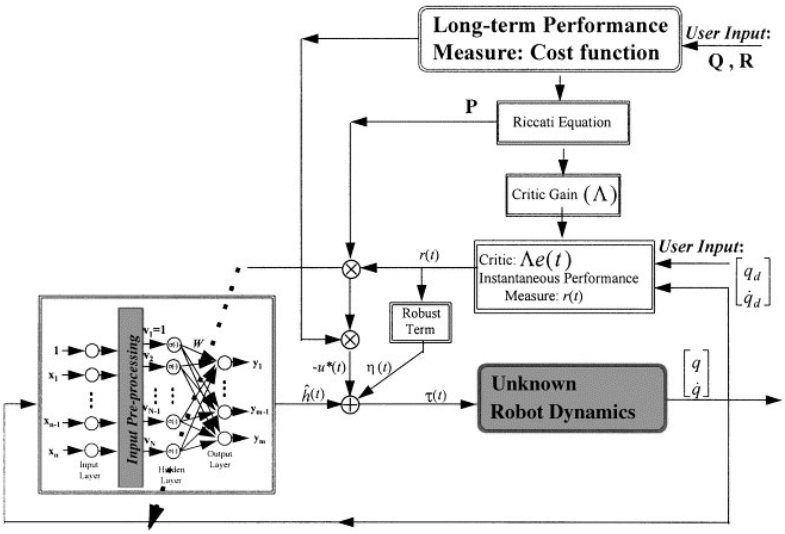
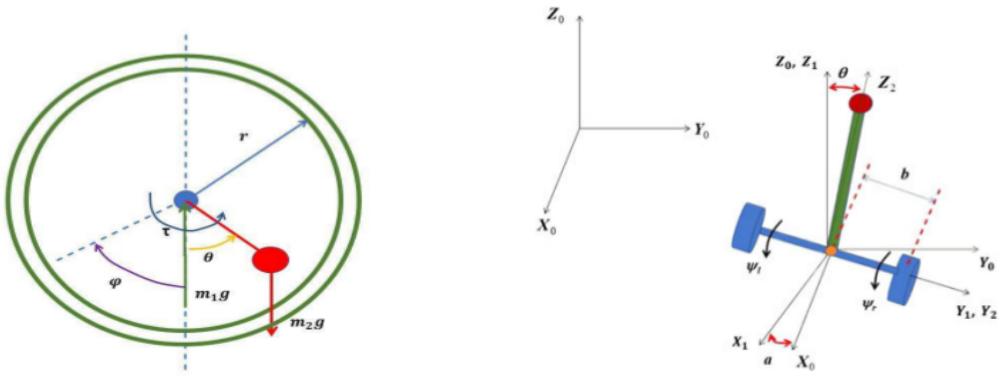


Figura 2.5: Control óptimo basado en redes neuronales. Fuente: `https://www.sciencedirect.com/science/article/pii/S0005109800000455`



(a) Robot esférico actuado por un péndulo. Fuente: [30]

(b) Diagrama de un péndulo invertido con ruedas. Fuente: [31]

Figura 2.6: Casos de aplicación de control óptimo a otros robots

Como parte del presente proyecto, se ha abordado el uso del control óptimo aplicado al control visual de robots manipuladores, en donde ha demostrado buenos resultados [32]. Además, se ha conseguido también ampliar su uso para el control de robots manipuladores móviles, aunando un control óptimo del brazo robótico junto con el control de la navegación de su base móvil [33].

2.3. Control visual

El concepto de **control visual**, comúnmente referido por su nombre en inglés, *visual servoing*, se emplea en la literatura para referirse al uso de técnicas de visión por computador para controlar el movimiento de un robot.

Uno de los primeros trabajos en los que este término fue acuñado es obra de Hill & Park [34], en el cual se estudia la utilización del control en bucle cerrado incluyendo información visual como parte de la realimentación. Sanderson & Weiss [35] establecen una clasificación de los sistemas de control visual en basados en posición y en imagen, la cual se ha asentado desde aquel momento.

Sin embargo, la evolución de este tipo de sistemas a partir de este momento fue lenta, con escasas contribuciones al respecto. Esto cambió hace poco más de una década, cuando los avances tanto en potencia de microprocesadores, como en calidad de cámaras y técnicas de procesado de imágenes provocaron un crecimiento importante en el número de autores interesados. En las siguientes referencias, se puede encontrar una revisión de las principales líneas de investigación dentro de los sistemas de control visual [36, 37, 38]. En la figura 2.7, se puede observar el esquema genérico de un sistema de control visual, mientras que en los próximos párrafos se describirán brevemente los distintos tipos de control visual existentes.

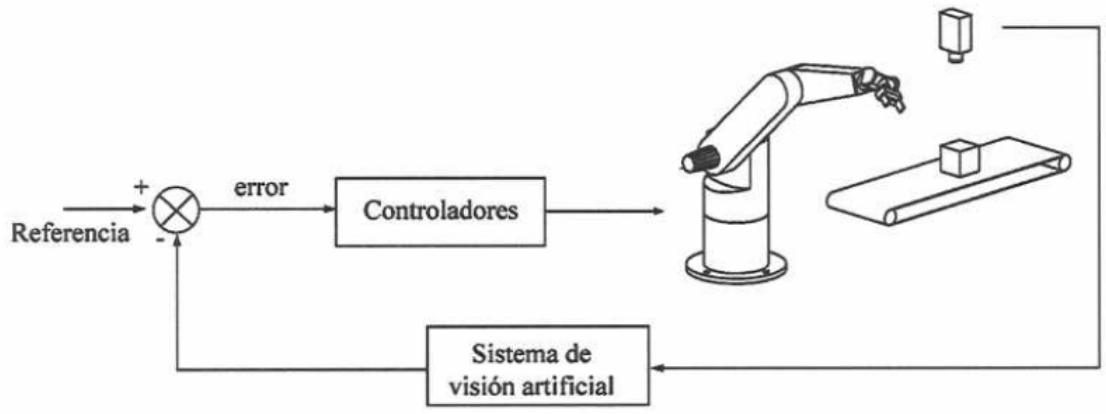


Figura 2.7: Esquema genérico de control visual. Fuente: [1]

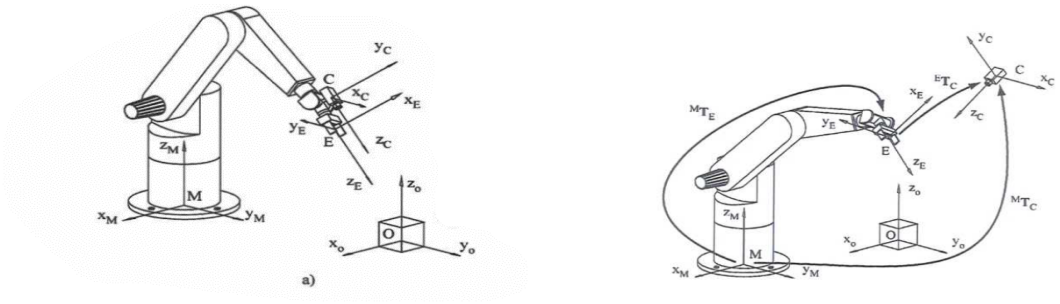
En la figura 2.7, aparecen una serie de componentes destacados de un control visual genérico, cuyo significado se explica a continuación:

- **Referencia**. Configuración final deseada para el robot. El tipo de referencia utilizada depende de si se emplea un controlador basado en posición (La referencia es una posición cartesiana en espacio 3D a alcanzar) o si se emplea un control basado en imagen (la referencia es una posición deseada de características visuales en la imagen).

- **Controlador.** Realiza el guiado del robot mediante el uso de la información visual, con el objetivo de alcanzar la referencia. El tipo de controlador depende de diversos factores, entre ellos si el control es basado en posición o en imagen, o bien si se trata de un control directo o indirecto.

- **Sistema de visión artificial**. Realimentación del sistema de control visual, encargado de la extracción de información visual del sistema que se necesita para guiar al robot. Si tratamos con un controlador basado en imagen, este simplemente extraerá la posición de las características visuales en la imagen. Sin embargo, si trabajamos con un control basado en posición, se deberá obtener la posición 3D del objeto en función de la información visual.

Además, otro dato importante a considerar en el campo del control visual es la ubicación del sistema de visión dentro de todo el conjunto. La información visual puede ser adquirida por una cámara colocada directamente en el extremo del robot manipulador, en cuyo caso al moverse el robot tendremos asociado un movimiento de la cámara. Por otro lado, la cámara puede estar fija en el espacio de trabajo del robot, de modo que sea capaz de observar su movimiento a partir de una posición constante. Ambas posibles configuraciones se representan en la figura 2.8, y toman el nombre de configuración *eye-in-hand* y *eye-to-hand* respectivamente.



(a) Configuración con cámara en el extremo del robot. Fuente: [1]

(b) Configuración con cámara externa al robot. Fuente: [1]

Figura 2.8: Posibles ubicaciones de la cámara en control visual.

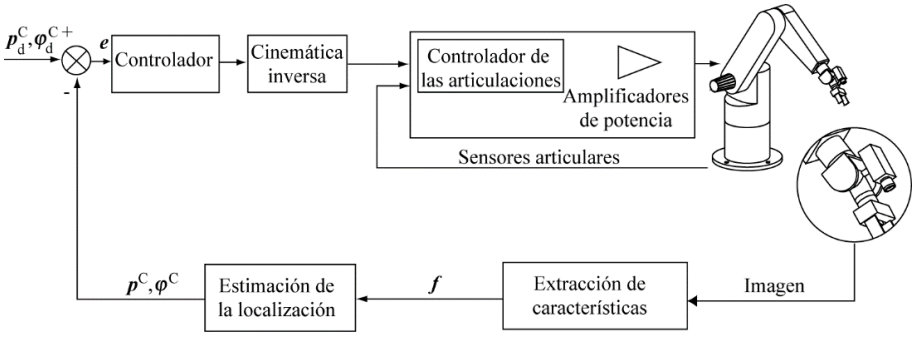
Como último apunte en lo que a tipología se refiere, además de según la posición de la cámara, o según el tipo de referencia con la que estemos trabajando (basado en posición/basado en imagen), existe otra característica a tener en cuenta a la hora de definir un controlador visual, y no es otro que el tipo de comando que emite el bucle de control. Siguiendo esta clasificación, podemos tener dos tipos de controladores: Directos e Indirectos.

En el control visual indirecto, la acción de control se obtiene como velocidades a aplicar al robot, suponiendo su dinámica despreciable, por lo que en estos casos se tienen en cuenta únicamente aspectos cinemáticos del robot. Por contra, en los controladores directos la acción de control se obtiene normalmente como pares a aplicar a las articulaciones, con las ventajas que ello conlleva, descritas en la sección 2.1.2.

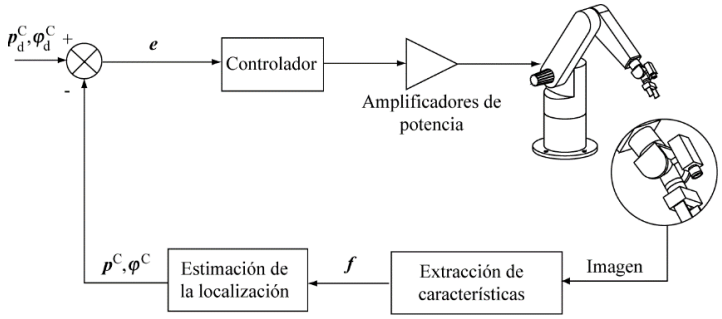
Así, una vez esclarecida la terminología en lo concerniente al control visual, podemos retomar el análisis del estado del arte en esta materia, dividiendo los trabajos existentes en función del tipo de control visual al que estén referidos.

En lo que refiere a los sistemas de control visual basado en posición, es adecuado citar tanto trabajos clásicos como [39] o [40], en el cual se estima de forma rápida mediante un método iterativo la posición de un objeto a partir de 4 puntos o más no coplanares, o a partir de 4 puntos coplanares en [41]. Cabe resaltar también investigaciones más recientes como la de [42] sobre sistemas estereoscópicos para su uso en controladores visuales basados en posición.

En la figura 2.9, se ilustran los dos posibles casos de un controlador visual basado en posición, en el que la acción de control es directa o indirecta.



(a) Control visual indirecto basado en posición. Fuente: [1]



(b) Control visual directo basado en posición. Fuente: [1]

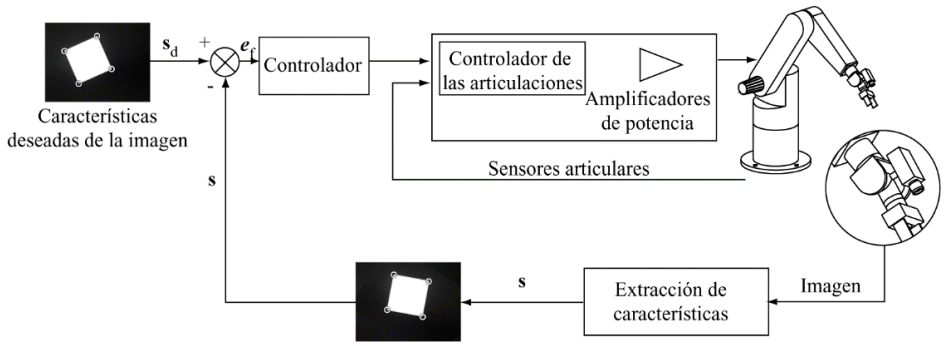
Figura 2.9: Tipos de controladores visuales basados en posición.

Por otro lado, en lo que a los controladores visuales basados en imagen se refiere, la literatura al respecto es muy extensa, sobre todo en el caso del control indirecto. Sin embargo, en los trabajos Chaumette & Hutchinson [36, 37] se puede encontrar una recopilación de los principales trabajos y problemáticas relacionadas con este tipo de controladores. En lo que respecta al control visual directo basado en imagen, existen algunos autores que han trabajado en ello con el fin de obtener mejores tiempos de respuesta y mejor estabilización frente a los controladores indirectos. Destaca en este aspecto el trabajo de Kelly [43], en el que considera la dinámica del manipulador para llevar a cabo el control directo haciendo uso de la teoría de Lyapunov para analizar la estabilidad. Por último, cabe destacar los trabajos de Pomares et al. [44], en el que se describe una estrategia de control visual en que las referencias visuales en la imagen dependen del tiempo, y con especial hincapié el trabajo de Cheah et al. [45], cuyo propósito es la implementación de un controlador visual adaptativo, que realice

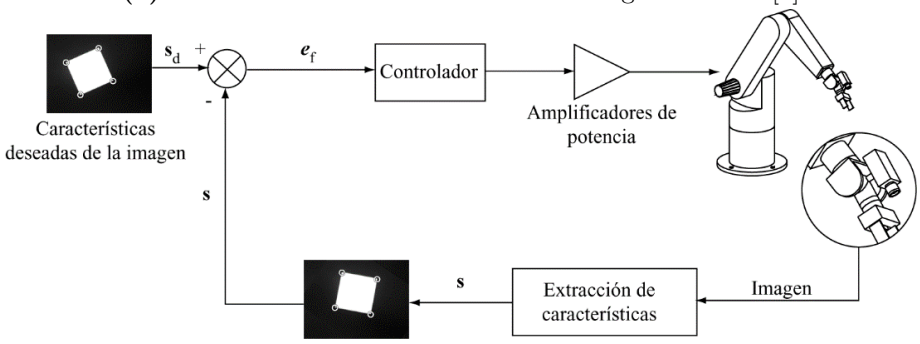
correctamente la tarea aun en presencia de errores en el modelo dinámico, o algunos parámetros de la tarea como la distancia entre la cámara y el objeto observado.

Así pues, el control visual basado en imagen sigue siendo una línea de investigación muy activa en la actualidad. En [46] se realiza un análisis de la estabilidad de un esquema de control visual aplicado a robots paralelos, con resultados robustos a perturbaciones. Otro trabajo interesante es el de Penin et al. [47], en donde se investiga la generación online de trayectorias óptimas para el seguimiento de objetivos con un drone mientras se satisfacen un conjunto de restricciones basadas en imagen. El objetivo de esta investigación es seguir de forma suave pero reactiva el objetivo móvil, evitando a la vez obstáculos en el entorno y oclusiones en el espacio imagen.

En la figura 2.10 se ilustran los dos posibles casos de un controlador visual basado en imagen, en el que la acción de control es directa o indirecta.



(a) Control visual indirecto basado en imagen. Fuente: [1]



(b) Control visual directo basado en imagen. Fuente: [1]

Figura 2.10: Tipos de controladores visuales basados en imagen.

3. Metodología: Bases teóricas y técnicas del desarrollo

En este capítulo, se van a exponer los fundamentos teóricos esenciales en los que se basan los controladores implementados en este proyecto, con el fin de justificar su empleo y validarlos matemáticamente.

Así pues, en primer lugar se desarrollarán teóricamente los controladores dinámicos multiarticulares implementados, indicando para cada uno de ellos su expresión matemática, ecuación en bucle cerrado, y demostración de equilibrio único.

En segundo lugar, se expondrán los controladores óptimos desarrollados en el marco de este proyecto, empezando por el controlador óptimo cartesiano, y siguiéndole el controlador visual óptimo basado en imagen. Para ambos, se mostrará el desarrollo para la obtención de su expresión matemática, y se explicarán los términos clave dentro del control óptimo de robots manipuladores, haciendo especial hincapié en el resultado que estos causan en el comportamiento del robot.

Por último, se mostrarán los detalles de la implementación de los controladores que se habrán desarrollado previamente, tanto a nivel del hardware empleado, como también en cuanto al software que se ha usado e implementado para conseguir plasmar dichos controladores.

3.1. Control dinámico multiarticular de robots manipuladores

Para empezar, considérese el modelo dinámico de un robot manipulador de n GDL expuesto en la ecuación (2.1), en el que se consideran condiciones ideales como que sus eslabones son rígidos, no existe fricción en sus uniones, y además sus accionadores son ideales. Este modelo en términos del vector de estado $\left[\boldsymbol{q}^T \dot{\boldsymbol{q}}^T\right]^T$ tomaría la siguiente forma:

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{bmatrix} = \begin{bmatrix}\dot{\boldsymbol{q}}\\ M(\boldsymbol{q})^{-1}\left[\boldsymbol{\tau}(t) - C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - G(\boldsymbol{q})\right]\end{bmatrix}$$

El problema de control de movimiento de robots manipuladores se puede formular del siguiente modo. Considérese la ecuación de la dinámica de un robot descrita en (2.1). Dado un conjunto de funciones vectoriales acotadas $\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d$ y $\ddot{\boldsymbol{q}}_d$ referidas como posiciones, velocidades y aceleraciones articulares *deseadas*, consiste en encontrar una función vectorial $\boldsymbol{\tau}$, de modo que las posiciones $\boldsymbol{q}$ asociadas a las coordenadas articulares del robot sigan de forma precisa a las establecidas por $\boldsymbol{q}_d$.

De manera más formal, el **objetivo del control de movimiento** consiste en calcular $\boldsymbol{\tau}$ de modo que:

$$\lim_{t\to\infty} \tilde{\boldsymbol{q}}(t) = \boldsymbol{0} \tag{3.1}$$

Donde $\tilde{\boldsymbol{q}} \in \Re^n$ denota el vector de errores de posiciones articulares, en adelante denominado error en posición, y obtenido como

$$\tilde{\boldsymbol{q}}(t) = \boldsymbol{q}_d(t) - \boldsymbol{q}(t).$$

En base a la definición anterior, es posible extenderla para que el vector $\dot{\tilde{\boldsymbol{q}}}(t) = \dot{\boldsymbol{q}}_d(t) - \dot{\boldsymbol{q}}(t)$ denote el error de velocidad articular. Así pues, si el objetivo de control se verifica, significará que las articulaciones del robot manipulador siguen asintóticamente la trayectoria de movimiento deseada.

El cálculo del vector $\boldsymbol{\tau}$ involucra en la mayoría de casos el uso de una función vectorial no lineal de $\boldsymbol{q}, \dot{\boldsymbol{q}}$ y $\ddot{\boldsymbol{q}}$. A esta función vectorial se la denomina *ley de control* o simplemente

controlador. Es conveniente recordar que los robots manipuladores habitualmente disponen de sensores de posición y velocidad para cada articulación, por lo cual los vectores $\boldsymbol{q}$ y $\dot{\boldsymbol{q}}$ son medibles y pueden emplearse en los controladores. Generalmente, la ley de control puede expresarse como una función de los siguientes términos expuestos en párrafos anteriores:

$$\boldsymbol{\tau} = \boldsymbol{\tau}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{q}_d, \dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d, M(\boldsymbol{q}), C(\boldsymbol{q}, \dot{\boldsymbol{q}}), G(\boldsymbol{q})\right) \tag{3.2}$$

Sin embargo, para fines prácticos es recomendable que el controlador no dependa de la aceleración articular $\ddot{\boldsymbol{q}}$, puesto que es un valor que no se suele poder medir fácilmente.

En la figura 3.1 se puede ver el diagrama de bloques genérico formado por un controlador en bucle cerrado con un robot.

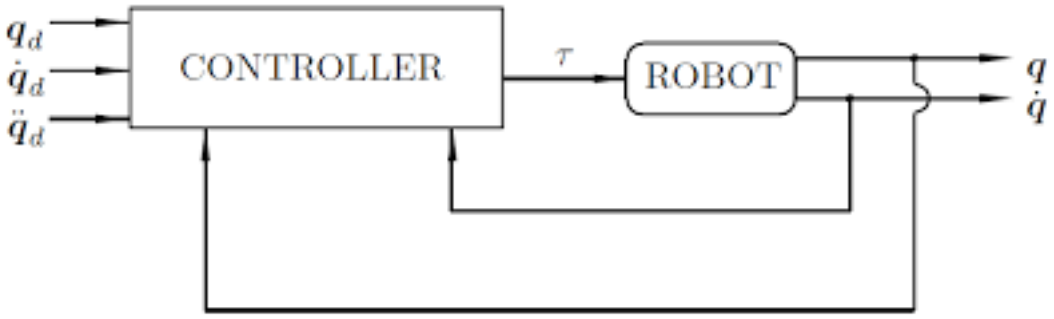


Figura 3.1: Control en bucle cerrado de robots. Fuente: [2]

Así pues, en las siguientes subsecciones de este capítulo se van a desarrollar teóricamente los controladores para el control de movimiento de robots manipuladores implementados en el marco de este proyecto. El contenido del análisis teórico que se va a tratar para cada controlador se puede resumir del siguiente modo:

1. Cálculo de la ecuación dinámica en bucle cerrado. Esto se obtiene mediante la sustitución de la acción de control $\boldsymbol{\tau}$ (3.2) en el modelo dinámico del robot (2.1). Normalmente, la ecuación en bucle cerrado resultante es una ecuación diferencial ordinaria no lineal.

2. Representación de la ecuación en bucle cerrado en espacio de estados. Dicha ecuación

tomará la forma:

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{q}_d - \boldsymbol{q} \\ \dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}\end{bmatrix} = \boldsymbol{f}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{q}_d, \dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d, M(\boldsymbol{q}), C(\boldsymbol{q}, \dot{\boldsymbol{q}}), G(\boldsymbol{q})\right).$$

Esta ecuación en bucle cerrado puede verse como un sistema dinámico, cuyas entradas son $\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d$ y $\ddot{\boldsymbol{q}}_d$, y siendo sus salidas el estado $\tilde{\boldsymbol{q}}(t) = \boldsymbol{q}_d(t) - \boldsymbol{q}(t)$ y $\dot{\tilde{\boldsymbol{q}}}(t) = \dot{\boldsymbol{q}}_d(t) - \dot{\boldsymbol{q}}(t)$.

3. Estudio de la existencia y posible unicidad de equilibrios de la ecuación en bucle cerrado obtenida en el apartado anterior. Con esto, se busca demostrar que el controlador tiene un único equilibrio, que satisface lo expuesto en (3.1).

Con esto, a continuación se plasma la base teórica de los controladores dinámicos multiarticulares implementados: *PD con prealimentación*, *PD+* y *Par-Calculado*.

3.1.1. Control PD con prealimentación

La implantación en la práctica de controladores para robots manipuladores se suele realizar haciendo uso de tecnología digital. El proceso habitual seguido por estos sistemas de control cuenta con las siguientes etapas:

- Muestreo de la posición articular $\boldsymbol{q}$ y de la velocidad $\dot{\boldsymbol{q}}$.
- Obtención de la acción de control $\boldsymbol{\tau}$ a partir de la ley de control.
- Envío de la acción de control a los actuadores.

En algunas aplicaciones donde se requiere que el robot lleve a cabo tareas repetitivas a gran velocidad, las etapas listadas anteriormente deben ejecutarse a una alta frecuencia. El *cuello de botella* en lo que a consumo de tiempo se refiere es el cálculo de la acción de control $\boldsymbol{\tau}$. Obviamente, una reducción en el tiempo de cálculo de $\boldsymbol{\tau}$ tendría como ventaja una mayor frecuencia de procesamiento, y en consecuencia un aumento en la capacidad del robot de realizar tareas más rápidamente.

Es por este motivo que surge el interés en controladores con relativamente poco poder de cálculo. Concretamente, este es el caso de los controladores que usan información basada en las posiciones, velocidades y aceleraciones articulares deseadas: $\boldsymbol{q}_d(t), \dot{\boldsymbol{q}}_d(t)$ y $\ddot{\boldsymbol{q}}_d(t)$. En estos casos, ya establecida la frecuencia de procesamiento, los términos de la ley de control que dependen exclusivamente de estos valores pueden ser calculados y almacenados en memoria, para ser leídos cuando sean necesarios durante el cálculo de la acción de control, reduciendo de esta forma el tiempo de cálculo.

Es por este interés práctico de incorporar el menor número posible de operaciones en tiempo real al implantar un controlador para robots que se propuso en la bibliografía especializada el **Control PD con prealimentación**, cuya ley de control viene dada por:

$$\boldsymbol{\tau} = K_p\tilde{\boldsymbol{q}} + K_v\dot{\tilde{\boldsymbol{q}}} + M(\boldsymbol{q}_d)\ddot{\boldsymbol{q}}_d + C(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)\dot{\boldsymbol{q}}_d + G(\boldsymbol{q}_d), \tag{3.3}$$

donde $K_p, K_v \in \Re^{n\times n}$ son matrices simétricas, definidas positivas, conocidas como ganancias de posición y velocidad, respectivamente. El teŕmino *prealimentación* en el nombre del controlador se debe al hecho de que en la ley de control se usa la dinámica del robot evaluada explícitamente en el movimiento deseado.

De igual forma, se supone que los eslabones del manipulador están unidos por articulaciones rotacionales, y que las cotas máximas de las normas de velocidad y aceleración deseadas, denotadas por $\|\dot{\boldsymbol{q}}_d\|_M$ y $\|\ddot{\boldsymbol{q}}_d\|_M$, son conocidas.

En la figura 3.2 se muestra el diagrama de bloque correspondiente a este controlador.

Los resultados obtenidos por diversos autores sobre el control de movimiento para robots mediante el control (3.3) han mostrado una excelente prestación, comparable hasta con el empleo de controladores del tipo Par-Calculado, el cual será expuesto más adelante en este capítulo. Sin embargo, estos resultados pueden ser engañosos, puesto que el buen comportamiento del controlador no es únicamente atribuible a la estructura del controlador en sí, sino también a una sintonización adecuada de la matriz de ganancia proporcional K_p.

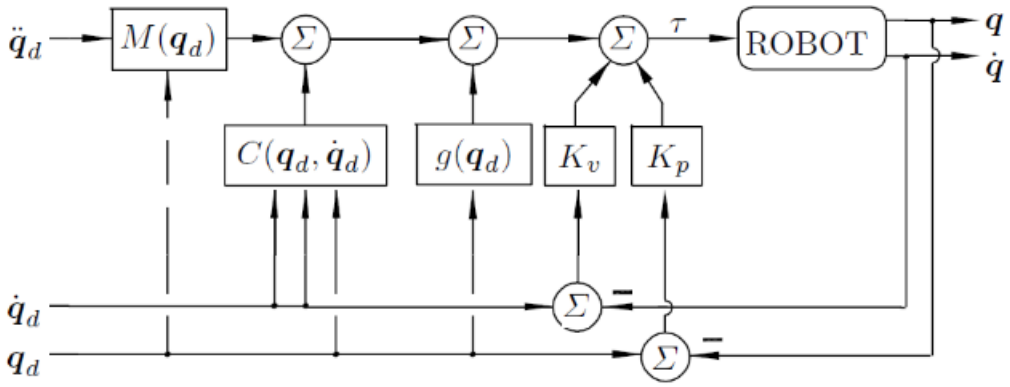


Figura 3.2: Control PD con prealimentación. Fuente: [2]

Así pues, el comportamiento en bucle cerrado del sistema se obtiene sustituyendo la acción de control $\boldsymbol{\tau}$ de (3.3) en el modelo dinámico del robot (2.1):

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + G(\boldsymbol{q}) = K_p\tilde{\boldsymbol{q}} + K_v\dot{\tilde{\boldsymbol{q}}} + M(\boldsymbol{q}_d)\ddot{\boldsymbol{q}}_d + C(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d)\dot{\boldsymbol{q}}_d + G(\boldsymbol{q}_d). \tag{3.4}$$

La ecuación en bucle cerrado (3.4) puede expresarse en términos del vector de estado $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T$ como

$$\frac{d}{dt}\begin{bmatrix}\tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix} = \begin{bmatrix}\dot{\tilde{\boldsymbol{q}}} \\ M(\boldsymbol{q})^{-1}\left[-K_p\tilde{\boldsymbol{q}} - K_v\dot{\tilde{\boldsymbol{q}}} - C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}} - \boldsymbol{h}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}})\right]\end{bmatrix}, \tag{3.5}$$

donde

$$\boldsymbol{h}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = \left[M(\boldsymbol{q}_d) - M(\boldsymbol{q})\right]\ddot{\boldsymbol{q}}_d + \left[C(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d) - C(\boldsymbol{q}, \dot{\boldsymbol{q}})\right]\dot{\boldsymbol{q}}_d + G(\boldsymbol{q}_d) - G(\boldsymbol{q}).$$

es la denominada dinámica residual, que se detalla en [48].

Es fácil demostrar que el origen $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T = 0 \in \Re^{2n}$ del espacio de estado es un equilibrio, independientemente de las matrices K_p y K_v. Sin embargo, la cantidad de equilibrios del sistema en bucle cerrado (3.5) depende del valor de la ganancia proporcional K_p. A continua-

ción, se van a presentar condiciones suficientes sobre K_p que garantizan la presencia de un único equilibrio, situado en el origen, para la ecuación en bucle cerrado del controlador (3.5).

Si se considera el caso de robots formados únicamente por articulaciones rotacionales, así como la selección suficientemente "grande" de K_p, puede demostrarse que el origen $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T = 0 \in \Re^{2n}$ es el único equilibrio de (3.5). Los equilibrios son los vectores constantes $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T = \left[\boldsymbol{s}^T \boldsymbol{0}^T\right]^T \in \Re^{2n}$, donde $\boldsymbol{s} \in \Re^n$ es una solución de

$$K_p \boldsymbol{s} + \boldsymbol{h}(\boldsymbol{s}, \boldsymbol{0}) = \boldsymbol{0}. \tag{3.6}$$

La expresión anterior siempre se cumple para la solución trivial $\boldsymbol{s} = \boldsymbol{0} \in \Re^n$, pero sería posible que otros vectores $\boldsymbol{s}$ pudiesen ser también soluciones de la ecuación, dependiendo del valor que tomase la ganancia proporcional K_p. En lo sucesivo, se van a indicar condiciones explícitas sobre dicha ganancia proporcional para asegurar la unicidad del equilibrio. A este efecto, defínase:

$$\boldsymbol{k}(\boldsymbol{s}) = K_p^{-1} \boldsymbol{h}(\boldsymbol{s}, \boldsymbol{0}).$$

El objetivo es notar que cualquier punto fijo $\boldsymbol{s} \in \Re^n$ de $\boldsymbol{k}(\boldsymbol{s})$ es una solución de (3.6). Por tanto, el interés será encontrar condiciones sobre K_p de modo que $\boldsymbol{k}(\boldsymbol{s})$ tenga un único punto fijo. Puesto que $\boldsymbol{s} = \boldsymbol{0}$ es siempre un punto fijo, entonces éste será el único.

Nótese que para todo vector $\boldsymbol{x}, \boldsymbol{y} \in \Re^n$, se tiene:

$$\begin{aligned} \|\boldsymbol{k}(\boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{y})\| &\leq \left\| K_p^{-1} \left[\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{0}) - \boldsymbol{h}(\boldsymbol{y}, \boldsymbol{0})\right] \right\|, \\ &\leq \lambda_{Max}\{K_p^{-1}\} \|\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{0}) - \boldsymbol{h}(\boldsymbol{y}, \boldsymbol{0})\|. \end{aligned}$$

Por otro lado, empleando la definición de la dinámica residual [48], se consigue llegar a:

$$\begin{aligned} \|\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{0}) - \boldsymbol{h}(\boldsymbol{y}, \boldsymbol{0})\| \leq & \left\| \left[M(\boldsymbol{q}_d - \boldsymbol{y}) - M(\boldsymbol{q}_d - x)\right] \ddot{\boldsymbol{q}}_d \right\| \\ & + \left\| \left[C(\boldsymbol{q}_d - y, \dot{\boldsymbol{q}}_d) - C(\boldsymbol{q_d} - \boldsymbol{x}, \dot{\boldsymbol{q}}_d)\right] \dot{\boldsymbol{q}}_d \right\| \\ & + \|G(\boldsymbol{q}_d - \boldsymbol{y}) - G(\boldsymbol{q}_d - \boldsymbol{x})\| \end{aligned}$$

Además, de las propiedades matemáticas de las distintas componentes del modelo dinámico

del robot (Matriz de Inercia, Matriz de Coriolis y Matriz de Gravedad), que se pueden extraer de [49], se desprende la existencia de constantes k_M, k_{C_1},k_{C_2} y k_G, tales que:

$$\begin{aligned}\|M(\boldsymbol{x})\boldsymbol{z} - M(\boldsymbol{y})\boldsymbol{z}\| &\leq k_M \|\boldsymbol{x}-\boldsymbol{y}\| \ \|\boldsymbol{z}\|,\\ \|C(\boldsymbol{x},\boldsymbol{z})\boldsymbol{w} - C(\boldsymbol{y},\boldsymbol{v})\boldsymbol{w}\| &\leq k_{C_1}\|\boldsymbol{z}-\boldsymbol{v}\| \ \|\boldsymbol{w}\| + k_{C_2}\|\boldsymbol{z}\| \ \|\boldsymbol{x}-\boldsymbol{y}\| \ \|\boldsymbol{w}\|,\\ \|G(\boldsymbol{x}) - G(\boldsymbol{y})\| &\leq k_G\|\boldsymbol{x}-\boldsymbol{y}\|,\end{aligned}$$

para todo $\boldsymbol{v},\boldsymbol{w},\boldsymbol{x},\boldsymbol{y},\boldsymbol{z} \in \Re^{\boldsymbol{n}}$. Teniendo en consideración esto último, se llega a la siguiente expresión:

$$\|\boldsymbol{h}(\boldsymbol{x},\boldsymbol{0}) - \boldsymbol{h}(\boldsymbol{y},\boldsymbol{0})\| \leq \left[k_G + k_M\|\ddot{\boldsymbol{q}}_d\|_M + k_{C_2}\|\dot{\boldsymbol{q}}_d\|_M^2\right] \ \|\boldsymbol{x}-\boldsymbol{y}\|.$$

A partir de esta expresión, usando $\lambda_{Max}\{K_p^{-1}\} = \frac{1}{\lambda_{min}\{K_p\}}$, dado que K_p es una matriz definida positiva simétrica, obtenemos:

$$\|\boldsymbol{k}(\boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{y})\| \leq \frac{1}{\lambda_{min}\{K_p\}}\left[k_G + k_M\|\ddot{\boldsymbol{q}}_d\|_M + k_{C_2}\|\dot{\boldsymbol{q}}_d\|_M^2\right] \ \|\boldsymbol{x}-\boldsymbol{y}\|.$$

Finalmente, en base a lo expuesto en el teorema de contracción de mapas [50], se afirma que:

$$\lambda_{min}\{K_p\} > k_G + k_M\|\ddot{\boldsymbol{q}}_d\|_M + k_{C_2}\|\dot{\boldsymbol{q}}_d\|_M^2 \tag{3.7}$$

es una condición suficiente para que $\boldsymbol{k}(\boldsymbol{s})$ tenga un único punto fijo, y en consecuencia, que el punto origen del espacio de estado sea el único equilibrio posible del sistema en bucle cerrado (3.5).

En conclusión, el control PD con prealimentación aplicado a robots manipuladores de n GDL puede, condicionado a usar matrices de diseño K_p y K_v suficientemente "grandes", satisfacer el objetivo del control de movimiento en forma global.

3.1.2. Control PD+

El objetivo de controlar el movimiento de robots manipuladores puede conseguirse de forma global haciendo uso del control PD con prealimentación expuesto en la sección 3.1.1. Sin embargo, a pesar de tratarse de una estrategia de control relativamente sencilla, requiere para su implementación el conocimiento del modelo dinámico del robot, aunque en realidad esta es una característica compartida de los controladores de movimiento. El problema en el caso del controlador PD con prealimentación radica en que también es necesario el uso del modelo dinámico en el procedimiento de sintonía, ya de por sí laborioso [2], y por si fuera poco se precisa de conocer de antemano la tarea que va a realizar el robot.

Así pues, a continuación se va a presentar otro de los controladores implementados, cuyas principales características son la garantía del cumplimiento de control de movimiento en forma global, así como un procedimiento trivial de sintonía de las ganancias: El control *PD+*.

El controlador **PD+** es uno de los controladores más sencillos que se pueden aplicar en el control multiarticular de seguimiento de trayectorias de un robot manipulador, con una garantía formal para el cumplimiento de este objetivo de control en forma global. La ley de control PD+ viene dada por la expresión siguiente:

$$\boldsymbol{\tau} = K_p\tilde{\boldsymbol{q}} + K_v\dot{\tilde{\boldsymbol{q}}} + M(\boldsymbol{q})\ddot{\boldsymbol{q}}_d + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_d + G(\boldsymbol{q}), \tag{3.8}$$

donde $K_p, K_v \in \Re^{n\times n}$ son de nuevo matrices simétricas definidas positivas a escoger por el diseñador del controlador, y como de costumbre $\tilde{\boldsymbol{q}} = \boldsymbol{q}_d - \boldsymbol{q}$ refiere al error de posición.

Al igual que en el caso del control PD con prealimentación, requiere conocer de forma precisa el modelo dinámico del manipulador, i.e., $M(\boldsymbol{q}), C(\boldsymbol{q},\dot{\boldsymbol{q}})$ y $G(\boldsymbol{q})$. Además, también será necesario conocer los valores de la trayectorias deseada $\boldsymbol{q}_d(t), \dot{\boldsymbol{q}}_d(t)$ y $\ddot{\boldsymbol{q}}_d(t)$ así como las mediciones $\boldsymbol{q}(t)$ y $\dot{\boldsymbol{q}}(t)$ del estado actual del robot.

No obstante, tal y como se mostrará más adelante, en contraposición al control PD con

prealimentación (3.3) en este caso la selección de K_p y K_v es completamente arbitraria, y no depende en absoluto ni de la tarea a realizar por el robot, ni del modelo dinámico del mismo. En la figura 3.3 se muestra el diagrama de bloques del control PD+ aplicado a robots manipuladores.

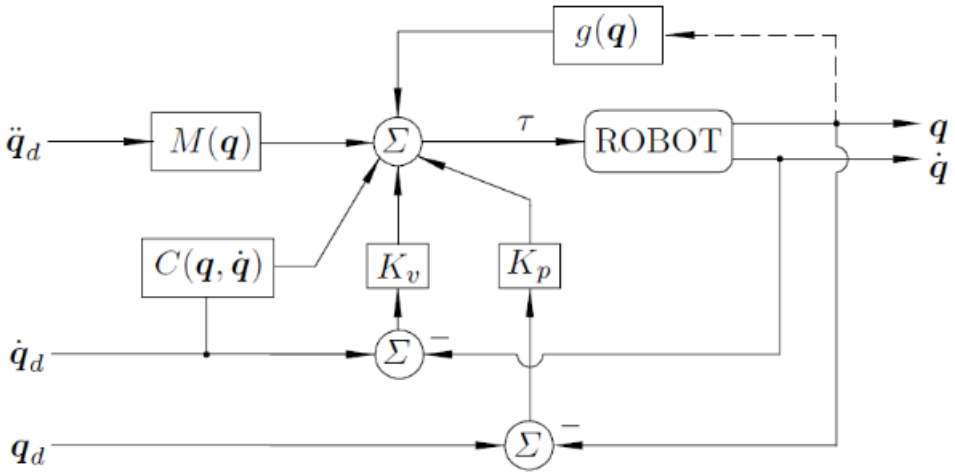


Figura 3.3: Control PD+. Fuente: [2]

La ecuación que describe el comportamiento del sistema en bucle cerrado se obtiene de nuevo al introducir la acción de control $\boldsymbol{\tau}$ de la ley de control (3.8) en el modelo dinámico del robot (2.1):

$$M(\boldsymbol{q})\ddot{\tilde{\boldsymbol{q}}} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}} = -K_p\tilde{\boldsymbol{q}} - K_v\dot{\tilde{\boldsymbol{q}}}.$$

Expresando esta ecuación en bucle cerrado en términos del vector de estado $\left[\tilde{\boldsymbol{q}}^T\dot{\tilde{\boldsymbol{q}}}^T\right]^T$, se tendría:

$$\frac{d}{dt}\begin{bmatrix}\tilde{\boldsymbol{q}}\\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix} = \begin{bmatrix}\dot{\tilde{\boldsymbol{q}}}\\ M(\boldsymbol{q}_d-\tilde{\boldsymbol{q}})^{-1}\left[-K_p\tilde{\boldsymbol{q}} - K_v\dot{\tilde{\boldsymbol{q}}} - C(\boldsymbol{q}_d-\tilde{\boldsymbol{q}},\dot{\boldsymbol{q}}_d-\dot{\tilde{\boldsymbol{q}}})\dot{\tilde{\boldsymbol{q}}}\right]\end{bmatrix}, \tag{3.9}$$

la cual es una ecuación diferencial no lineal y no autónoma. Esta última propiedad es debida a que la ecuación depende de forma explícita de las funciones del tiempo $\boldsymbol{q}_d(t)$ y $\dot{\boldsymbol{q}}_d(t)$.

Por otro lado, cabe destacar además que la ecuación en bucle cerrado tiene como único estado de equilibrio al origen $\left[\tilde{\boldsymbol{q}}^T\dot{\tilde{\boldsymbol{q}}}^T\right]^T = \mathbf{0} \in \Re^{2n}$. Por lo cual, en el caso de que $\boldsymbol{q}(0) = \boldsymbol{q}_d(0)$

y $\dot{\boldsymbol{q}}(0) = \dot{\boldsymbol{q}}_d(0)$, entonces $\boldsymbol{q}(t) = \boldsymbol{q}_d(t)$ y $\dot{\boldsymbol{q}}(t) = \dot{\boldsymbol{q}}_d(t)\ \forall t \geq 0$.

Nótese que lo anterior se puede concluir tan solo a partir del concepto de equilibrio sin necesidad de invocar ningún otro análisis. Por contra, para obtener conclusiones si no se cumple la igualdad en el estado inicial del robot tanto en posición como en velocidad con la trayectoria planificada, es necesario realizar el análisis de estabilidad del equilibrio, mediante el estudio de la estabilidad de Lyapunov [51].

Con el fin de analizar la estabilidad en el origen, se va a considerar la siguiente función candidata de Lyapunov:

$$\begin{aligned} V(t, \tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) &= \frac{1}{2} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} K_p & 0 \\ 0 & M(\boldsymbol{q}_d - \tilde{\boldsymbol{q}}) \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}, \\ &= \frac{1}{2} \dot{\tilde{\boldsymbol{q}}}^T M(\boldsymbol{q}) \dot{\tilde{\boldsymbol{q}}} + \frac{1}{2} \tilde{\boldsymbol{q}}^T K_p \tilde{\boldsymbol{q}}. \end{aligned} \tag{3.10}$$

Como se puede apreciar, la función es definida positiva puesto que tanto la matriz de inercia $M(\boldsymbol{q})$ como la matriz de ganancia proporcional K_p son definidas positivas. Así pues, derivando (3.10) con respecto al tiempo, se obtiene:

$$\dot{V}(t, \tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = \dot{\tilde{\boldsymbol{q}}}^T M(\boldsymbol{q}) \ddot{\tilde{\boldsymbol{q}}} + \frac{1}{2} \dot{\tilde{\boldsymbol{q}}}^T \dot{M}(\boldsymbol{q}) \dot{\tilde{\boldsymbol{q}}} + \tilde{\boldsymbol{q}}^T K_p \dot{\tilde{\boldsymbol{q}}}.$$

Aislando en la ecuación en bucle cerrado (3.9) el término $M(\boldsymbol{q})\ddot{\boldsymbol{q}}$ y sustituyendo la expresión resultante en la ecuación anterior, se obtiene que:

$$\begin{aligned} \dot{V}(t, \tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) &= -\dot{\tilde{\boldsymbol{q}}}^T K_v \dot{\tilde{\boldsymbol{q}}}, \\ &= - \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & K_v \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix} \leq 0 \end{aligned} \tag{3.11}$$

donde el término $\dot{\tilde{\boldsymbol{q}}}^T \left[\frac{1}{2}\dot{M} - C\right] \dot{\tilde{\boldsymbol{q}}}$ ha sido eliminado por una de las propiedades de la matriz de Coriolis expuestas en [48], que indica una relación entre las matrices de Inercia y Coriolis de la forma $\dot{\boldsymbol{q}}^T \left[\frac{1}{2}\dot{M} - C\right] \dot{\boldsymbol{q}} = 0\ \forall \boldsymbol{q}, \dot{\boldsymbol{q}} \in \Re^n$.

De forma similar, por el teorema de estabilidad y acotamiento de soluciones enmarcado en la teoría de estabilidad de Lyapunov [51], se concluye inmediatamente estabilidad del origen $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T = \mathbf{0} \in \Re^{2n}$. Además, los errores de posición y velocidad están acotados, i.e.,

$$\tilde{\boldsymbol{q}},\ \dot{\tilde{\boldsymbol{q}}} \in L_\infty^n. \tag{3.12}$$

Por otro lado, empleando el Lema 2.2 de [2], en el que se trata el valor de la cota de la norma euclidiana de una función vectorial temporal, se obtiene que el error de velocidad pertenece al espacio de funciones L_2^n, i.e.,

$$\dot{\tilde{\boldsymbol{q}}} \in L_2^n. \tag{3.13}$$

Considerando las conclusiones previas, se va a proceder a demostrar que el error en velocidad $\dot{\tilde{\boldsymbol{q}}}$ tiende asintóticamente a cero. Con este objetivo, nótese de la ecuación en bucle cerrado (3.9) que:

$$\ddot{\tilde{\boldsymbol{q}}} = M(\boldsymbol{q})^{-1}\left[-K_p\tilde{\boldsymbol{q}} - K_v\dot{\tilde{\boldsymbol{q}}} - C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\tilde{\boldsymbol{q}}}\right]$$

en donde el lado derecho de la expresión sería acotado como consecuencia de (3.12), siempre y cuando los argumentos de $M(\boldsymbol{q})$ y $C(\boldsymbol{q},\dot{\boldsymbol{q}})$ también lo sean. En esta circunstancia, el error en aceleración, $\ddot{\tilde{\boldsymbol{q}}}$ es un vector acotado, i.e.,

$$\ddot{\tilde{\boldsymbol{q}}} \in L_\infty^n.$$

Este último hecho, unido a (3.13) y al mencionado Lema 2.2, implica que:

$$\lim_{t\to\infty} \dot{\tilde{\boldsymbol{q}}}(t) = \lim_{t\to\infty} (\dot{\boldsymbol{q}}_d(t) - \dot{\boldsymbol{q}}(t)) = \mathbf{0} \in \Re^n.$$

De este modo, se demuestra la estabilidad asintótica del error en velocidad de este controlador.

En conclusión, para cualquier valor de las matrices simétricas definidas positivas K_p y K_v, el origen de la ecuación en bucle cerrado del control PD+ expresado en términos del vec-

tor de estado $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T$, es asintóticamente estable globalmente. Por tanto, el control PD+ cumple con el objetivo de control de movimiento en forma global. Debido a esto, se tiene que para cualquier error de posición inicial $\tilde{\boldsymbol{q}}(0) \in \Re^n$ y de velocidad $\dot{\tilde{\boldsymbol{q}}}(0) \in \Re^n$, se tiene que $\lim_{t\to\infty} \tilde{\boldsymbol{q}}(t) = \mathbf{0}$.

3.1.3. Control Par-Calculado

Los controladores presentados en las subsecciones anteriores tienen en común en sus leyes de control la presencia explícita de un controlador lineal del tipo Proporcional-Derivativo (PD). Como excepción a esta característica, en este subapartado se va a presentar un controlador de movimiento cuya ley de control no presenta explícitamente este término lineal PD. El controlador al que se está haciendo referencia no es otro que el *Control Par-Calculado*.

El control Par-Calculado permite obtener una ecuación en bucle cerrado lineal en términos de las variables de estado, lo cual no sucedía ni en el control PD con prealimentación ni en el control PD+. Además, este controlador satisface el objetivo del control de movimiento con una selección trivial de sus parámetros de diseño, lo cual lo hace muy atractivo. A continuación, se detallarán estas peculiaridades como se ha venido haciendo en los casos anteriores.

El modelo dinámico (2.1) que caracteriza el comportamiento de los manipuladores es habitualmente no lineal en términos de las variables de estado (posiciones y velocidades articulares). Este hecho podría inducir a la idea de que en cualquier controlador, la ecuación diferencial resultante de obtener su expresión en bucle cerrado debería ser no lineal de igual forma en las variables de estado mencionadas.

Esta idea intuitiva se confirma para los controladores estudiados previamente en este capítulo. Sin embargo, existe un controlador que igualmente es no lineal en cuanto a sus variables de estado, pero con el que se consigue describir el sistema de control en bucle cerrado mediante una ecuación diferencial lineal. Con este controlador se satisface el objetivo de control de movimiento de forma global, con un proceso de selección de sus parámetros de diseño

completamente arbitrario. Se trata del control **Par-Calculado**.

La ley de control correspondiente al control Par-Calculado viene dada por:

$$\boldsymbol{\tau} = M(\boldsymbol{q}) \left[\ddot{\boldsymbol{q}}_d + K_v \dot{\tilde{\boldsymbol{q}}} + K_p \tilde{\boldsymbol{q}} \right] + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} + G(\boldsymbol{q}), \tag{3.14}$$

donde K_v y K_p son matrices simétricas definidas positivas de diseño, y $\tilde{\boldsymbol{q}}$ se refiere una vez más al error en posición.

En este caso, a pesar de contar de nuevo con la presencia de una expresión del tipo PD, como es el término $K_v \dot{\tilde{\boldsymbol{q}}} + K_p \tilde{\boldsymbol{q}}$ en la ley de control (3.14), éstos son realmente multiplicados por la matriz de inercia $M(\boldsymbol{q}_d - \tilde{\boldsymbol{q}})$. Este detalle tiene como resultado que efectivamente la expresión del controlador cuente con un término del tipo PD, pero éste no es un controlador lineal, ya que las ganancias proporcional y derivativa no son constantes, sino que dependen explícitamente del error en posición $\tilde{\boldsymbol{q}}$. Este hecho se observa más claramente expresando la ley de control (3.14) de la siguiente forma:

$$\boldsymbol{\tau} = M(\boldsymbol{q}) \ddot{\boldsymbol{q}}_d + M(\boldsymbol{q}_d - \tilde{\boldsymbol{q}}) K_v \dot{\tilde{\boldsymbol{q}}} + M(\boldsymbol{q}_d - \tilde{\boldsymbol{q}}) K_p \tilde{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} + G(\boldsymbol{q}).$$

El control Par-Calculado fue una de las primeras estructuras de control de movimiento basadas en el modelo del manipulador, en otras palabras, que hace uso explícito del conocimiento de las matrices $M(\boldsymbol{q})$, $C(\boldsymbol{q}, \dot{\boldsymbol{q}})$ y $G(\boldsymbol{q})$. También cabe destacar el hecho de que tanto la trayectoria de movimiento deseada: $\boldsymbol{q}_d(t)$, $\dot{\boldsymbol{q}}_d(t)$ y $\ddot{\boldsymbol{q}}_d(t)$, así como las medidas de la posición actual $\boldsymbol{q}(t)$ y de la velocidad $\dot{\boldsymbol{q}}(t)$, se emplean para calcular la acción de control (3.14). El diagrama de bloques correspondiente al control Par-Calculado de un robot manipulador se ilustra en la figura 3.4.

La ecuación en bucle cerrado se consigue de nuevo sustituyendo la acción de control $\boldsymbol{\tau}$ de la ley de control (3.14) en la ecuación del modelo dinámico del robot (2.1):

$$M(\boldsymbol{q}) \ddot{\boldsymbol{q}} = M(\boldsymbol{q}) \left[\ddot{\boldsymbol{q}}_d + K_v \dot{\tilde{\boldsymbol{q}}} + K_p \tilde{\boldsymbol{q}} \right]. \tag{3.15}$$

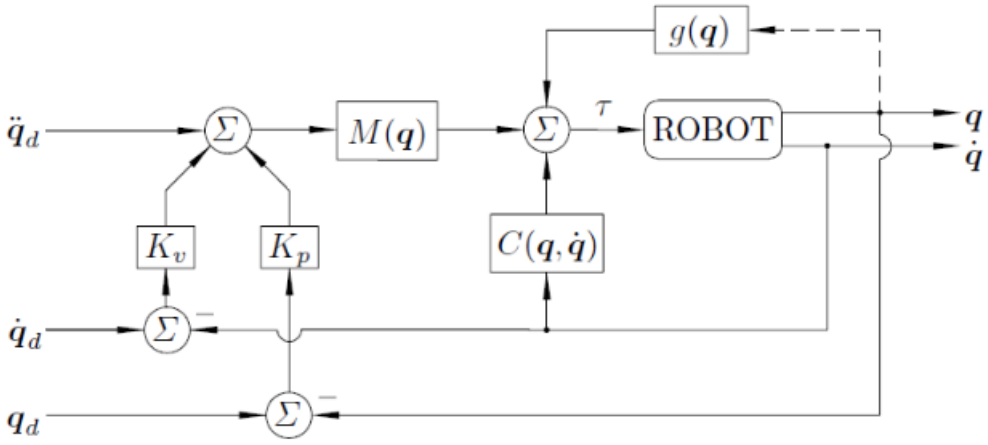


Figura 3.4: Control Par-Calculado. Fuente: [2]

De nuevo, si se echa un vistazo a las propiedades de los parámetros dinámicos del robot expuestos en [48], se observa que $M(\boldsymbol{q})$ es una matriz definida positiva, y por lo tanto también invertible, luego la ecuación (3.15) se reduce a:

$$\ddot{\tilde{\boldsymbol{q}}} + K_v\dot{\tilde{\boldsymbol{q}}} + K_p\tilde{\boldsymbol{q}} = \mathbf{0},$$

que a su vez, expresada en términos del vector de estado $\left[\tilde{\boldsymbol{q}}^T\dot{\tilde{\boldsymbol{q}}}^T\right]^T$, toma la forma:

$$\begin{aligned}\frac{d}{dt}\begin{bmatrix}\tilde{\boldsymbol{q}}\\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix} &= \begin{bmatrix}\dot{\tilde{\boldsymbol{q}}}\\ -K_p\tilde{\boldsymbol{q}} - K_v\dot{\tilde{\boldsymbol{q}}}\end{bmatrix},\\ &= \begin{bmatrix}0 & I\\ -K_p & -K_v\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{q}}\\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix},\end{aligned} \tag{3.16}$$

donde I es la matriz identidad de dimensión n.

Es interesante observar como efectivamente la ecuación en bucle cerrado (3.16) representa una ecuación lineal y autónoma, cuyo único estado de equilibrio es $\left[\tilde{\boldsymbol{q}}^T\dot{\tilde{\boldsymbol{q}}}^T\right]^T = \mathbf{0} \in \Re^{2n}$. Esta unicidad del equilibrio es consecuencia de que la matriz K_p es por diseño definida positiva, y por lo tanto, no singular.

Así pues, dado que la expresión del controlador en bucle cerrado (3.16) es lineal y autónoma, las soluciones de ésta pueden obtenerse en forma cerrada. No obstante, a continuación se realizará el estudio de estabilidad del origen como equilibrio de la ecuación en bucle cerrado.

En primer lugar, defínase la constante ε como:

$$\lambda_{min}\{K_v\} > \varepsilon > 0.$$

Multiplicando por $\boldsymbol{x}^T\boldsymbol{x}$ donde $\boldsymbol{x} \in \Re^n$ es cualquier vector no nulo, se llega a $\lambda_{min}\{K_v\}\boldsymbol{x}^T\boldsymbol{x} > \varepsilon\boldsymbol{x}^T\boldsymbol{x}$. Así, teniendo en cuenta que K_v es por diseño una matriz simétrica, entonces $\boldsymbol{x}^T K_v \boldsymbol{x} \geq \lambda_{min}\{K_v\}\boldsymbol{x}^T\boldsymbol{x}$, y en consecuencia:

$$\boldsymbol{x}^T \left[K_v - \varepsilon I\right] \boldsymbol{x} > 0 \qquad\qquad \forall \boldsymbol{x} \neq \mathbf{0} \in \Re^n,$$

lo cual significa que la matriz $K_v - \varepsilon I$ es definida positiva, i.e.,

$$K_v - \varepsilon I > 0. \tag{3.17}$$

Teniendo presente el desarrollo anterior, así como la positividad de la matriz K_p y de la constante ε, se puede concluir que:

$$K_p + \varepsilon K_v - \varepsilon^2 I > 0. \tag{3.18}$$

Pasando ahora al análisis de estabilidad en el equilibrio, ubicado en el origen, de la ecuación en bucle cerrado (3.16), considérese la siguiente función candidata de Lyapunov:

$$\begin{aligned} V(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) &= \frac{1}{2} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} K_p + \varepsilon K_v & \varepsilon I \\ \varepsilon I & I \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}, \\ &= \frac{1}{2} \left[\dot{\tilde{\boldsymbol{q}}} + \varepsilon\tilde{\boldsymbol{q}}\right]^T \left[\dot{\tilde{\boldsymbol{q}}} + \varepsilon\tilde{\boldsymbol{q}}\right] + \frac{1}{2}\tilde{\boldsymbol{q}}^T \left[K_p + \varepsilon K_v - \varepsilon^2 I\right] \tilde{\boldsymbol{q}}, \end{aligned} \tag{3.19}$$

donde la constante ε satisface tanto (3.17) como por supuesto (3.18). Se puede observar muy claramente como la función de Lyapunov escogida (3.19) es definida positiva en forma

global y radialmente desacotada.

Se puede expresar también la función candidata de Lyapunov $V(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}})$ en (3.19) de la siguiente forma:

$$V(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = \frac{1}{2}\dot{\tilde{\boldsymbol{q}}}^T \dot{\tilde{\boldsymbol{q}}} + \frac{1}{2}\tilde{\boldsymbol{q}}^T \left[K_p + \varepsilon K_v\right] \tilde{\boldsymbol{q}} + \varepsilon \tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}},$$

cuya derivada con respecto al tiempo es

$$\dot{V}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = \ddot{\tilde{\boldsymbol{q}}}^T \dot{\tilde{\boldsymbol{q}}} + \tilde{\boldsymbol{q}}^T \left[K_p + \varepsilon K_v\right] \dot{\tilde{\boldsymbol{q}}} + \varepsilon \dot{\tilde{\boldsymbol{q}}}^T \dot{\tilde{\boldsymbol{q}}} + \varepsilon \tilde{\boldsymbol{q}}^T \ddot{\tilde{\boldsymbol{q}}}.$$

Así, si se sustituye $\ddot{\tilde{\boldsymbol{q}}}$ de la ecuación en bucle cerrado (3.16) en la expresión anterior, y se opera para simplificar todo, el resultado es:

$$\begin{aligned} \dot{V}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) &= -\dot{\tilde{\boldsymbol{q}}}^T \left[K_v - \varepsilon I\right] \dot{\tilde{\boldsymbol{q}}} - \varepsilon \tilde{\boldsymbol{q}}^T K_p \tilde{\boldsymbol{q}}, \\ &= -\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} \varepsilon K_p & 0 \\ 0 & K_v - \varepsilon I \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}. \end{aligned} \tag{3.20}$$

Como ε se escoge de modo que $K_v - \varepsilon I > 0$, y teniendo en cuenta que también K_p es por diseño definida positiva, se puede afirmar que la función $\dot{V}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}})$ en (3.20) es una función definida negativa en forma global. Así pues, se ha encontrado una función candidata que cumple con la teoría de la estabilidad de Lyapunov, por lo que se puede concluir que el origen $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T = \mathbf{0} \in \Re^{2n}$ de la ecuación en bucle cerrado es asintóticamente estable de forma global y en consecuencia:

$$\lim_{t\to\infty} \dot{\tilde{\boldsymbol{q}}}(t) = \mathbf{0},$$

$$\lim_{t\to\infty} \tilde{\boldsymbol{q}}(t) = \mathbf{0},$$

lo cual significa la verificación del cumplimiento del objetivo de control de movimiento.

Con fines de implantación práctica, las matrices de diseño K_p y K_v pueden ser diagonales, por lo cual la ecuación en bucle cerrado (3.16) representa un sistema lineal multivariable desacoplado. En otras palabras, el comportamiento dinámico de los errores de posición de cada articulación viene dado por ecuaciones diferenciales lineales de segundo orden, donde

cada una de ellas es independiente del resto. En esta tesitura, la elección de las matrices K_p y K_v puede hacerse específicamente como:

$$K_p = diag\{\omega_1^2, \ldots, \omega_n^2\},$$
$$K_v = diag\{2\omega_1, \ldots, 2\omega_n\}.$$

Escogiendo estos valores para las ganancias proporcional y derivativa, cada unión responde igual que un sistema lineal de segundo orden críticamente amortiguado, que constaría de un ancho de banda ω_i, el cual denota la velocidad de respuesta de la articulación y, por tanto, la tasa de decrecimiento exponencial de los errores $\tilde{\boldsymbol{q}}(t)$ y $\dot{\tilde{\boldsymbol{q}}}(t)$.

En conclusión, en este subapartado se ha analizado el control Par-Calculado, justificando de forma teórica una serie de propiedades. Así, se tiene que para cualquier selección de las matrices simétricas de diseño definidas positivas K_p y K_v, el origen de la ecuación en bucle cerrado del controlador expresado en términos del vector de estado $\left[\tilde{\boldsymbol{q}}^T \dot{\tilde{\boldsymbol{q}}}^T\right]^T$ es estable asintóticamente de forma global. En consecuencia, el control Par-Calculado satisface el objetivo del control de movimiento globalmente, por lo que para cualquier error inicial de posición $\tilde{\boldsymbol{q}}(0) \in \Re^n$ y de velocidad $\dot{\tilde{\boldsymbol{q}}}(0) \in \Re^n$, se tiene que $\lim_{t\to\infty} \tilde{\boldsymbol{q}}(t) = \boldsymbol{0}$.

3.1.4. Control cartesiano basado en Par-Calculado

Una vez estudiados los controladores en espacio articular implementados en el proyecto, en este apartado se va a introducir de forma muy breve cómo aplicar la técnica del control Par-Calculado para llevar a cabo el seguimiento de trayectorias en el espacio cartesiano.

Como se puede suponer, las modificaciones a realizar en el controlador vendrán dadas por el hecho de que tanto la trayectoria a seguir como los valores actuales de posición y velocidad del robot manipulador se nos presentarán como posiciones y velocidades en el espacio cartesiano. Por tanto, para poder aplicar la base teórica de este controlador, se debe previamente convertir los valores en el espacio cartesiano a magnitudes articulares.

Para este fin, se necesita de la aplicación de la cinemática diferencial, que en el caso de un robot manipulador viene dada por la matriz Jacobiana. Esta matriz relaciona las velocidades articulares del robot con la velocidad lineal de su extremo, permitiendo así obtener una equivalencia entre ambos espacios de coordenadas. De forma genérica, la expresión analítica de la matriz Jacobiana toma la forma:

$$J(\boldsymbol{q}) = \begin{pmatrix} \frac{\partial f_x}{\partial q_1} & \cdots & \frac{\partial f_x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_\gamma}{\partial q_1} & \cdots & \frac{\partial f_\gamma}{\partial q_n} \end{pmatrix} \tag{3.21}$$

donde cada fila representa la variación de una coordenada en el espacio cartesiano $(x, y, z, \alpha, \beta, \gamma)$ en función de los valores de las coordenadas articulares del robot manipulador. Así pues, con la cinemática diferencial se establecen las siguientes relaciones:

$$\begin{aligned} \dot{\boldsymbol{x}} &= J(\boldsymbol{q})\boldsymbol{q}, \\ \dot{\boldsymbol{q}} &= J^{+}(\boldsymbol{q})\dot{\boldsymbol{x}}. \end{aligned} \tag{3.22}$$

En las relaciones mostradas en (3.22), $\dot{\boldsymbol{x}}$ representa la velocidad en espacio cartesiano del extremo del robot manipulador. $\dot{\boldsymbol{q}}$ denota como de costumbre el vector de velocidades articulares de cada articulación del robot, $J(\boldsymbol{q})$ es la Jacobiana del robot en una determinada posición $\boldsymbol{q}$, y por último $J^{+}(\boldsymbol{q})$ es la pseudo-inversa de la matriz Jacobiana del robot. Se hace uso de esta notación con el fin de generalizar lo máximo posible la expresión anterior. Dado que no es el objetivo del proyecto, no se ahondará en mayor detalle en este tema, pudiendo encontrar información más detallada en lo que respecta al cálculo y justificación de la cinemática diferencial en bibliografía de referencia como [9, 15].

Así pues, esclarecido el uso de la matriz Jacobiana para cambiar entre espacio articular y cartesiano, se retomará la obtención del controlador cartesiano basado en Par-Calculado. La ley de control $\boldsymbol{\tau}$ que define este controlador viene dada por:

$$\boldsymbol{\tau} = M(\boldsymbol{q})J^{+}\left[\ddot{\boldsymbol{x}}_d + K_v\dot{\tilde{\boldsymbol{x}}} + K_p\tilde{\boldsymbol{x}} - \dot{J}\dot{\boldsymbol{q}}\right] + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + G(\boldsymbol{q}), \tag{3.23}$$

donde $K_p, K_v \in \Re^{6\times6}$ son las ganancias propocional y derivativa, de dimensión 6×6 al estar actuando sobre vectores de coordenadas cartesianas. $M(\boldsymbol{q})$, $C(\boldsymbol{q},\dot{\boldsymbol{q}})$ y $G(\boldsymbol{q})$ son los parámetros del modelo dinámico del robot (2.1), como de costumbre. Por su parte $\tilde{\boldsymbol{x}} = \boldsymbol{x}_d - \boldsymbol{x}$ denota el error en posición cartesiana del extremo del robot, mientras que $\dot{\tilde{\boldsymbol{x}}}$ es el error en velocidad cartesiana. Además, J^+ es la pseudo-inversa de la matriz Jacobiana del robot (3.21), y $\dot{J}$ la derivada con respecto al tiempo de dicha matriz.

Como se puede apreciar en la ley de control (3.23), se trata de un controlador prácticamente idéntico al del caso articular (3.14), pero cambiando las variables articulares por cartesianas, y añadiendo las versiones pertinentes de la matriz Jacobiana para realizar el cambio de nuevo a coordenadas cartesianas, para así poder obtener la acción de control $\boldsymbol{\tau}$ a aplicar a las articulaciones del robot.

Así pues, ssustituyendo en la ecuación del modelo dinámico (2.1) la expresión de la ley de control, se tiene el comportamiento en bucle cerrado del sistema de control, que viene dado por la expresión:

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} = M(\boldsymbol{q})J^+ \left[\ddot{\boldsymbol{x}}_d + K_v\dot{\tilde{\boldsymbol{x}}} + K_p\tilde{\boldsymbol{x}} - \dot{J}\dot{\boldsymbol{q}}\right]. \tag{3.24}$$

La ecuación del controlador en bucle cerrado obtenida (3.24) puede reducirse para finalmente obtener que:

$$\ddot{\tilde{\boldsymbol{x}}} = -K_v\dot{\tilde{\boldsymbol{x}}} - K_p\tilde{\boldsymbol{x}},$$

de donde se puede extraer que la dinámica del error es independiente de la configuración articular. Con esto, extendiéndolo a todo el desarrollo expuesto en el apartado anterior para el control Par-Calculado en espacio articular, se concluye que el controlador cumple con el objetivo de control de movimiento de forma global.

En conclusión, este controlador es tan solo una extensión a espacio cartesiano del indicado en (3.14). Este cambio viene motivado por lo intuitivo de llevar a cabo la planificación de la tarea en espacio cartesiano, ya que a los seres humanos nos es mucho más sencillo establecer posiciones en el espacio cartesiano que en base a unas coordenadas articulares.

3.2. Control óptimo

En esta sección, se va a exponer el desarrollo teórico de los controladores óptimos diseñados e implementados para este proyecto, haciendo especial hincapié en los efectos y beneficios del control óptimo de robots manipuladores.

Así pues, en primer lugar se va a exponer el diseño del controlador óptimo cartesiano implementado, indicando las etapas en su desarrollo y los distintos efectos que se pueden conseguir según se modifique el valor del parámetro clave del control óptimo.

Tras esto, en segundo lugar se abordará el desarrollo del controlador visual óptimo basado en imagen que se ha diseñado en el marco de este proyecto. Para ello, además de las bases matemáticas propias del controlador óptimo, se denotarán también los aspectos básicos necesarios para llevar a cabo un control visual basado en imagen.

Antes de empezar, cabe recordar brevemente el concepto de control óptimo aplicado a robots manipuladores, que ya se introdujo en el capítulo 2.

En el control óptimo de robots manipuladores, se trata de aprovechar la redundancia de este tipo de robots, considerando en el proceso la dinámica del robot. De este modo, mediante esta técnica se puede conseguir la optimización de las señales motoras o pares articulares enviados al sistema mecánico durante la ejecución de la tarea, al tratar de conseguir la mejor configuración articular posible para llevar a cabo el seguimiento de la trayectoria deseada, a la vez que se trata de satisfacer una restricción dada al controlador a través de una determinada matriz de pesos.

Sin más, en adelante se van a definir los controladores diseñados e implementados a lo largo de este proyecto.

3.2.1. Control óptimo cartesiano

Para entender el desarrollo detrás de este controlador, es necesario realizar un breve repaso de algunos de los términos vistos anteriormente en este proyecto.

En primer lugar, dado que se trata de un controlador cartesiano, es importante recordar el concepto de cinemática diferencial de un robot manipulador, el cual establece la relación entre las velocidades articulares del robot y la correspondiente velocidad del efector final, a través del uso de la matriz Jacobiana (3.22).

En cuanto a la dinámica del robot, se ha insistido mucho a lo largo de este proyecto en la importancia del modelo dinámico para el control de movimiento (2.1). Sin embargo, con el objetivo de simplificar futuras ecuaciones, se va a redefinir el modelo dinámico del robot como:

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{\tau} + F_{cg}(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{3.25}$$

donde $F_{cg}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -C(\boldsymbol{q}, \dot{\boldsymbol{q}}) - G(\boldsymbol{q})$ representa una contracción de las matrices de Coriolis y gravedad en un único término.

El uso del modelo dinámico de un robot manipulador ha sido usado en muchas aproximaciones de técnicas de control de movimiento. Sin embargo, la propuesta por Udwadia [24] dió una nueva perspectiva en lo referente al seguimiento de trayectorias con control óptimo para sistemas mecánicos no lineales. Básicamente, el esquema de control propuesto por Udwadia supone un sistema con m restricciones (holonomicas y/o no holonomicas) que representan la tarea a describir por el robot. La derivada respecto al tiempo de dichas restricciones viene dada por la ecuación:

$$A(\boldsymbol{q}, \dot{\boldsymbol{q}}, t)\ddot{\boldsymbol{q}} = \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \tag{3.26}$$

donde $A(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \Re^{m \times n}$ y $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) \in \Re^{m \times 1}$ son la matriz y el vector obtenidos, respectivamente. El controlador óptimo trata de minimizar los pares de control $\boldsymbol{\tau}$ del sistema mecánico,

a la vez que se lleva a cabo una determinada tarea considerando la siguiente función de coste:

$$\boldsymbol{\Omega}(t) = \boldsymbol{\tau}^T \boldsymbol{W}(t) \boldsymbol{\tau} \tag{3.27}$$

donde $\boldsymbol{W}(t)$ es una matriz de pesos dependiente del tiempo. La función de control que minimiza $\boldsymbol{\Omega}(t)$ del sistema mecánico basado en el modelo dinámico expresado en (3.25) mientras realiza la tarea descrita en la ecuación (3.26) viene dada por (por claridad en la expresión, se omiten las dependencias del tiempo o las posiciones articulares):

$$\boldsymbol{\tau} = \boldsymbol{W}^{-\frac{1}{2}} \left(A M^{-1} \boldsymbol{W}^{-\frac{1}{2}} \right)^{+} \left(\boldsymbol{b} - A M^{-1} F_{cg} \right), \tag{3.28}$$

donde M es la matriz de inercia, y el superíndice + denota la pseudo-inversa de una matriz cualquiera. Como se puede apreciar en la ecuación (3.28), la matriz $\boldsymbol{W}$ es la variable clave en la ley de control, ya que determina cómo la acción de control se va a distribuir en las articulaciones.

Por otro lado, en el caso de que se desee que el extremo del robot siga una trayectoria cartesiana, el controlador deberá cumplir la restricción dada por la ecuación:

$$(\ddot{\boldsymbol{x}}_d - \ddot{\boldsymbol{x}}) + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}), \tag{3.29}$$

donde $K_p, K_v \in \Re^{6\times 6}$ son matrices simétricas definidas positivas, que como de costumbre denotan las ganancias proporcional y derivativa respectivamente. Por su parte, $\ddot{\boldsymbol{x}}, \dot{\boldsymbol{x}}$ y $\boldsymbol{x}$ representan la aceleración, velocidad y posición del extremo del robot en el espacio cartesiano, y añadiendo el subíndice d, hacemos referencia a las mismas magnitudes en la trayectoria deseada para el robot.

Así pues, teniendo en cuenta la definición de la cinemática directa diferencial (3.22), derivando respecto al tiempo la primera expresión, $\dot{\boldsymbol{x}} = J(\boldsymbol{q})\dot{\boldsymbol{q}}$, se tiene que:

$$\ddot{\boldsymbol{x}} = J(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}(\boldsymbol{q})\dot{\boldsymbol{q}}.$$

Sustituyendo esta expresión en (3.29) y operando, se obtiene que:

$$\ddot{\boldsymbol{x}}_d + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}) = J\ddot{\boldsymbol{q}} + \dot{J}\dot{\boldsymbol{q}}. \tag{3.30}$$

Esta última ecuación (3.30) puede expresarse en los términos descritos en la ecuación (3.26) de la siguiente manera:

$$\begin{aligned} A(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= J(\boldsymbol{q}), \\ \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \ddot{\boldsymbol{x}}_d + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}) - \dot{J}\dot{\boldsymbol{q}}. \end{aligned} \tag{3.31}$$

Así, sustituyendo los términos A y $\boldsymbol{b}$ obtenidos en (3.31) dentro de la expresión genérica del control óptimo representada en (3.28), y utilizando distintos valores para la matriz de pesos $\boldsymbol{W}$, se pueden obtener los distintos comportamientos asociados a este controlador.

Con esto, una vez expuesto el desarrollo para la obtención del controlador óptimo cartesiano, a continuación se va a exponer la demostración matemática de la estabilidad de dicho controlador. Para ello, considérese la ley de control genérica del controlador óptimo expresada en (3.28).

Dados los valores de A y $\boldsymbol{b}$ obtenidos en (3.31), si se sustituyen en (3.28) la ley de control resultante será la siguiente:

$$\boldsymbol{\tau} = \boldsymbol{W}^{-\frac{1}{2}} \left(JM^{-1}\boldsymbol{W}^{-\frac{1}{2}} \right)^{+} \left(\ddot{\boldsymbol{x}}_d + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}) - \dot{J}\dot{\boldsymbol{q}} - JM^{-1}F_{cg} \right), \tag{3.32}$$

donde todos los términos han sido descritos en párrafos anteriores. Así pues, el comportamiento en bucle cerrado del sistema, resultante de sustituir la ley de control $\boldsymbol{\tau}$ (3.32) en la expresión reducida del modelo dinámico (3.25) será:

$$M\ddot{\boldsymbol{q}} - F_{cg} = \boldsymbol{W}^{-\frac{1}{2}} \left(JM^{-1}\boldsymbol{W}^{-\frac{1}{2}} \right)^{+} \left(\ddot{\boldsymbol{x}}_d + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}) - \dot{J}\dot{\boldsymbol{q}} - JM^{-1}F_{cg} \right). \tag{3.33}$$

Así pues, multiplicando toda la ecuación (3.33) por el término $\left(JM^{-1}\boldsymbol{W}^{-\frac{1}{2}}\right)\boldsymbol{W}^{\frac{1}{2}}$, se tiene:

$$J\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{x}}_d + K_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + K_p(\boldsymbol{x}_d - \boldsymbol{x}) - \dot{J}\dot{\boldsymbol{q}}.$$

Por último, reordenando términos en la expresión anterior, y considerando el error en posición cartesiana como $\tilde{\boldsymbol{x}} = \boldsymbol{x}_d - \boldsymbol{x}$, y de igual forma para velocidad y aceleración, se llega finalmente a la siguiente expresión:

$$\ddot{\tilde{\boldsymbol{x}}} = -K_v\dot{\tilde{\boldsymbol{x}}} - K_p\tilde{\boldsymbol{x}}, \tag{3.34}$$

de donde se puede concluir que el controlador realiza el seguimiento de la trayectoria con un decrecimiento asintótico de la función de error (3.34).

Con esto, se concluye por tanto que el controlador óptimo cartesiano desarrollado cumple con el objetivo de control de movimiento globalmente, teniéndose una estabilidad asintótica en la función de error como se ha demostrado. En el capítulo 4, se mostrará cómo afectan distintos valores de la matriz $\boldsymbol{W}$ al desempeño del robot.

3.2.2. Control visual óptimo

En esta subsección, se va a describir la formulación de un sistema de control visual directo basado en imagen, dedicado al seguimiento de trayectorias con optimización del esfuerzo articular, que ha sido desarrollado en el marco de este proyecto. Como siempre a lo largo de este capítulo, se van a describir los componentes más importantes de esta estrategia de control aplicada al control de movimiento de robots manipuladores.

De igual modo que en el caso del control óptimo cartesiano, haciendo uso de esta ley de control adaptada al control visual directo basado en imagen es posible tanto optimizar los pares aplicados a las articulaciones en el seguimiento de la trayectoria, como también redistribuir el esfuerzo ejercido en cada articulación, especificando así cuáles soportarán más esfuerzo durante el desarrollo de la tarea.

Así pues, en primer lugar es necesario definir una serie de conceptos importantes en el campo del control visual, empezando por definir los vectores $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ como la posición, velocidad y aceleración articular del robot, como de costumbre.

Tal y como se ha indicado anteriormente, el robot será guiado mediante el uso de información visual. Generalmente, esta información visual es un vector de k puntos característicos en el espacio imagen, representados por el vector $\boldsymbol{s} = [f_{1x}, f_{1y}, f_{2x}, f_{2y}, \cdots, f_{kx}, f_{ky}]^T \in \Re^{2k}$. La matriz de interacción, cuya expresión se puede consultar en [1] establece la siguiente relación entre el espacio imagen y el espacio cartesiano 3D:

$$\dot{\boldsymbol{s}} = L_s(\boldsymbol{s}, \boldsymbol{Z})\dot{\boldsymbol{x}}, \tag{3.35}$$

donde $\dot{\boldsymbol{s}}$ es la derivada con respecto al tiempo de las características visuales, y $\boldsymbol{x}, \dot{\boldsymbol{x}}$ representan como de costumbre la posición y velocidad cartesiana del extremo del robot. Para este desarrollo, se va a considerar una configuración *eye-in-hand*, descrita en la página 18, y por tanto la posición y velocidad cartesianas de la cámara coincidirán con las del extremo del robot.

Por su parte, la Jacobiana del robot relaciona la velocidad del efector final del robot con la velocidad de sus articulaciones, tal y como se indica en (3.22). Así pues, combinando esta relación con la ecuación (3.35), se obtiene la relación entre la velocidad articular del robot y la de las características visuales:

$$\dot{\boldsymbol{s}} = L_s(\boldsymbol{s}, \boldsymbol{Z})J(\boldsymbol{q})\dot{\boldsymbol{q}} = L_J(\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{Z})\dot{\boldsymbol{q}}, \tag{3.36}$$

donde $L_J = L_s(\boldsymbol{s}, \boldsymbol{Z})J(\boldsymbol{q}) = L_J(\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{Z}) \in \Re^{2k \times n}$.

Así pues, derivando la ecuación (3.36) con respecto al tiempo, se tiene (de nuevo, se omiten

las dependencias de las variables para obtener una notación más compacta):

$$\ddot{\boldsymbol{s}} = L_J \ddot{\boldsymbol{q}} + \dot{L}_J \dot{\boldsymbol{q}}, \tag{3.37}$$

donde $\ddot{\boldsymbol{s}}$ representa la aceleración de las características en el espacio imagen.

Queda por definir la referencia en el espacio imagen, $\ddot{\boldsymbol{s}}_r$, que permita desarrollar correctamente el seguimiento de las características visuales. Para ello, se considera:

$$\ddot{\boldsymbol{s}}_r = \ddot{\boldsymbol{s}}_d + K_v(\dot{\boldsymbol{s}}_d - \dot{\boldsymbol{s}}) + K_p(\boldsymbol{s}_d - \boldsymbol{s}) = \ddot{\boldsymbol{s}}_d + K_v \dot{\tilde{\boldsymbol{s}}} + K_p \tilde{\boldsymbol{s}}, \tag{3.38}$$

donde K_p y K_v son matrices PD simétricas definidas positivas. $\ddot{\boldsymbol{s}}_d, \dot{\boldsymbol{s}}_d, \boldsymbol{s}_d$ son las aceleraciones, velocidades y posiciones deseadas para las características de la imagen, y finalmente $\dot{\tilde{\boldsymbol{s}}} = \dot{\boldsymbol{s}}_d - \dot{\boldsymbol{s}}$ y $\tilde{\boldsymbol{s}} = \boldsymbol{s}_d - \boldsymbol{s}$ denotan los errores en velocidad y posición de las características en la imagen, respectivamente.

Una vez esclarecidos estos términos básicos, a continuación se va a extender el sistema de control visual a un control visual directo, generando así los pares articulares necesarios para el seguimiento, antes de pasar a desarrollar a partir de ello el control óptimo. Para ello, considérese el modelo dinámico del robot manipulador (2.1). A partir de la combinación de las ecuaciones (2.1), (3.37) y (3.38) el controlador visual directo que se obtiene es el siguiente:

$$\boldsymbol{\tau} = M(\boldsymbol{q}) L_J^+ \left(\ddot{\boldsymbol{s}}_d + K_v \dot{\tilde{\boldsymbol{s}}} + K_p \tilde{\boldsymbol{s}} - \dot{L}_J \dot{\boldsymbol{q}} \right) + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + G(\boldsymbol{q}). \tag{3.39}$$

Considerando las suposiciones previas, sustituyendo la ley de control (3.39) en el modelo dinámico del robot (2.1), se obtiene la ecuación en bucle cerrado:

$$M(\boldsymbol{q}) \ddot{\boldsymbol{q}} = M(\boldsymbol{q}) L_J^+ \left(\ddot{\boldsymbol{s}}_d + K_v \dot{\tilde{\boldsymbol{s}}} + K_p \tilde{\boldsymbol{s}} - \dot{L}_J \dot{\boldsymbol{q}} \right). \tag{3.40}$$

Así pues, si pre-multiplicando ambos lados de la ecuación (3.40) por el término $L_J \left(M(\boldsymbol{q})\right)^{-1}$, se obtiene:

$$L_J \ddot{\boldsymbol{q}} = \ddot{\boldsymbol{s}}_r - \dot{L}_J \dot{\boldsymbol{q}} \rightarrow L_J \ddot{\boldsymbol{q}} + \dot{L}_J \dot{\boldsymbol{q}} = \ddot{\boldsymbol{s}}_r.$$

Finalmente, considerando de nuevo la ecuación (3.37), la parte izquierda de la ecuación anterior equivale a $\ddot{\boldsymbol{s}}$, y por la ecuación (3.38), la parte derecha equivale a $\ddot{\boldsymbol{s}}_d + K_v\dot{\tilde{\boldsymbol{s}}} + K_p\tilde{\boldsymbol{s}}$. Por tanto, tras operar se tiene que:

$$\ddot{\tilde{\boldsymbol{s}}} = -K_v\dot{\tilde{\boldsymbol{s}}} - K_p\tilde{\boldsymbol{s}}. \tag{3.41}$$

Gracias a la ecuación (3.41), se puede afirmar que con este controlador conseguimos un decrecimiento asintótico del error en el espacio imagen. Este hecho se extiende cuando ampliamos este control a un control óptimo, como se va a hacer a continuación.

Como recordatorio, la ley de control que minimiza los pares aplicados a un robot manipulador mientras ejecuta el seguimiento de una determinada referencia viene dado en la ecuación (3.28), mientras que las restricciones de la tarea se especifican mediante la relación (3.26).

Así pues, el principal objetivo a conseguir es guiar el robot a lo largo de una trayectoria definida en el espacio imagen. Por tanto, en este caso la descripción de la tarea viene dada por la ecuación:

$$(\ddot{\boldsymbol{s}}_d - \ddot{\boldsymbol{s}}) + K_v(\dot{\boldsymbol{s}}_d - \dot{\boldsymbol{s}}) + K_p(\boldsymbol{s}_d - \boldsymbol{s}) = 0$$

Esta ecuación se puede expresar respecto al error en imagen. A partir de las ecuaciones (3.38) y (3.37) se puede conseguir:

$$\ddot{\boldsymbol{s}}_d + K_v\dot{\tilde{\boldsymbol{s}}} + K_p\tilde{\boldsymbol{s}} - \dot{L}_J\dot{\boldsymbol{q}} = L_J\ddot{\boldsymbol{q}}. \tag{3.42}$$

De este modo, ya se pueden expresar las restricciones de la tarea indicadas en (3.42) en la forma necesaria para el control óptimo (3.26), siendo:

$$\begin{aligned} A &= L_J, \\ \boldsymbol{b} &= \ddot{\boldsymbol{s}}_d + K_v\dot{\tilde{\boldsymbol{s}}} + K_p\tilde{\boldsymbol{s}} - \dot{L}_J\dot{\boldsymbol{q}}. \end{aligned} \tag{3.43}$$

Por último, reemplazando con los valores de (3.43) en la ley de control (3.28), se obtiene

finalmente el controlador visual óptimo:

$$\boldsymbol{\tau} = \boldsymbol{W}^{-\frac{1}{2}} \left(L_J M^{-1} \boldsymbol{W}^{-\frac{1}{2}} \right)^{+} \left(\ddot{\boldsymbol{s}}_d + K_v \dot{\tilde{\boldsymbol{s}}} + K_p \tilde{\boldsymbol{s}} - \dot{L}_J \dot{\boldsymbol{q}} - L_J M^{-1} F_{cg} \right), \tag{3.44}$$

Este controlador no pudo ser implementado en el robot TIAGo debido a restricciones en el hardware del robot, y en menor medida al hecho de no contar el brazo del robot con una cámara en su extremo, y ser su integración bastante complicada.

Sin embargo, su desempeño sí que fue probado en una serie de simulaciones llevadas a cabo para la publicación derivada de esta parte del proyecto [32]. En ellas, se puede observar como efectivamente en función del valor de la matriz de pesos W se consiguen comportamientos distintos del manipulador. Además, este desarrollo se extendió para el caso de robots manipuladores móviles, como se puede ver en la publicación derivada de este estudio [33].

Con esto, termina el desarrollo teórico de los controladores diseñados e implementados a lo largo de este proyecto, y se pasará precisamente a analizar los detalles de esta implementación en lo que resta de este capítulo.

3.3. Implementación técnica

En esta sección del proyecto, se van a exponer los detalles de la implementación de los controladores estudiados en los puntos anteriores.

Para ello, en primer lugar se hará un repaso al robot escogido para la implantación práctica de dichos controladores: el robot **TIAGo**, de la compañía *PAL Robotics*, líder mundial en robots humanoides y colaborativos. Así pues, se pondrán de manifiesto las características *hardware* y *software* principales del robot, haciendo especial hincapié en las partes que más han influido en el desarrollo de este proyecto, y obviando algunas que no hayan tenido ningún papel en él.

Tras esto, en la segunda parte de esta sección, se profundizará en el *software* usado y desa-

rrollado para la implementación de los controladores dinámicos multiarticulares estudiados. En este punto, se describirá también el reto de trasladar a un robot real tan complejo como TIAGo los controladores probados previamente en su propia simulación.

En definitiva, el objetivo de esta sección es tratar el paso de la teoría a la práctica, en el que todo el desarrollo teórico llevado a cabo anteriormente toma forma mediante una implementación concreta, y se prueba en un robot real de altas prestaciones, como es TIAGo.

3.3.1. Hardware: Robot TIAGo

TIAGo (*Take It And Go*) es un robot manipulador móvil de servicio colaborativo creado por la compañía PAL Robotics, y cuya primera versión data del 2015. Debido a sus características, es una plataforma ideal para su uso en el ámbito investigador, especialmente en los campos de la robótica asistencial en hogares, o incluso para la industria *ligera*. Esto se debe a que combina diversos campos destacados de la robótica como son la movilidad autónoma, percepción, manipulación e Interacción Humano - Máquina (HMI) en una única plataforma de coste relativamente reducido. En la figura 3.5, se presenta un resumen de los componentes principales de TIAGo.

Figura 3.5: *Hardware* principal de TIAGo. Fuente: `http://tiago.pal-robotics.com/`

Este robot tiene una altura de entre 110 y 145 cm (dependiendo de la extensión del torso prismático), pesa 72 kg y su base móvil tiene un diámetro de 54 cm, lo cual lo hace un robot de tamaño medio-alto dentro de su sector.

En cuanto a los GDL, la base móvil cuenta con un sistema diferencial de 2 ruedas, el torso está formado por una articulación prismática, y la cabeza por dos articulaciones que permiten el "giro de cuello" y el cabeceo frontal. Sin embargo, la parte que más interesa para el proyecto es el brazo manipulador de 7 GDL (4 GDL del brazo y 3 de la muñeca). Más adelante se hará especial hincapié en las características del manipulador al ser realmente donde se van a implantar los controladores dinámicos multiarticulares.

La plataforma está dotada a su vez de multitud de sensores, desde su base móvil que cuenta con un LIDAR, diversos Sonars y una IMU, sensores de corriente en los motores del brazo, y una cámara RGB-D en la cabeza.

PAL Robotics proporciona un manual detallado de las características del robot TIAGo, sin embargo en este proyecto se mencionarán tan solo aquellas que hayan intervenido en algún momento en la implantación de los controladores. Por ello, se dejarán de lado algunas partes como la base móvil o el torso, entre otros.

En lo que refiere a la capacidad computacional del robot, en la versión usada para las pruebas que se detallarán en el capítulo 4, contaba con una CPU *Intel i7 Haswell*, con 16 GB de memoria RAM, un disco duro sólido de 512 GB, y conectividad tanto por Wi-Fi como Bluetooth.

Así, en este punto se llega al componente *hardware* más importante del robot en el ámbito de este proyecto: el manipulador.

El brazo manipulador del robot TIAGo está compuesto por cuatro módulos M90 y una muñeca de 3 GDL M3D, tal y como se muestra en la figura 3.6

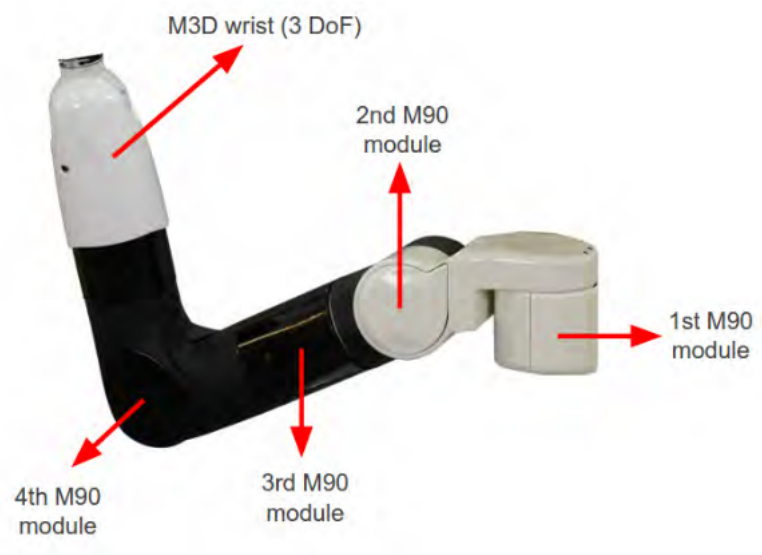


Figura 3.6: Componentes del manipulador de TIAGo. Fuente: [3]

Como se puede observar, este manipulador consta de 7 GDL. Sin embargo, desgraciadamente solo las 4 articulaciones del brazo permiten un control directo, mientras que los GDL propios de la muñeca tan solo permiten un control en posición a alto nivel. Por este motivo, los controladores diseñados se han implantado únicamente a las 4 primeras articulaciones del manipulador, las cuales si bien es cierto que en simulación sí que permiten un control directo, en el robot real lo que se permite es el control de corriente, por lo que se deberá corregir la acción de control a enviar a cada actuador considerando para ello las reductoras empleadas, así como la constante de par propia de cada motor. Este hecho se tratará más adelante en el capítulo 4.

Siguiendo con las características del manipulador, cabe destacar que tiene un peso de 10 kg, y que puede soportar una carga de unos 3 kg en muñeca (sin efector final). En la tabla 3.1, se resumen el resto de propiedades importantes de las articulaciones del manipulador.

Por último en lo que a especificaciones *hardware* se refiere, otro componente importante del robot es la cámara RGB-D montada en la cabeza de TIAGo, y que proporciona imágenes RGB junto a una imagen de profundidad obtenida usando un proyector de infrarrojos y una cámara infrarroja. Esta imagen de profundidad se usa para obtener la nube de puntos de la

Description	Reduction	Max. speed [rpm]	Nominal torque [Nm]
1st module	100:1	18	39
2nd module	100:1	18	39
3rd module	100:1	22	22
4th module	100:1	22	22

Tabla 3.1: Especificaciones del brazo de TIAGo [3]

escena. Así, las especificaciones de esta cámara se muestran en la tabla 3.2.

Fabricante	Orbbec
Modelo	Astra
Campo de visión	60º H, 49.5º V 73º D
Interfaz	USB 2.0
Transmisión de la imagen en color	QVGA 320x240 @ 30 fps, VGA 640x480 @ 30 fps
Transmisión de la imagen de profundidad	QVGA 320x240 @ 30 fps, VGA 640x480 @ 30 fps
Rango del sensor de profundidad	0.6 - 8 m.

Tabla 3.2: Especificaciones cámara RGB-D de TIAGo [3]

Con esto, quedan descritas las características *hardware* más importantes del robot TIAGo en el marco de este proyecto.

En el próximo apartado, se detallará la arquitectura software del robot, haciendo especial hincapié en lo referente al diseño e implantación de controladores, así como en las librerías y paquetes de software adicionales que han sido necesarios para la implementación del software deseado.

3.3.2. Software: ROS Control. Diseño de controladores e integración en la estructura software de TIAGo

En esta subsección, se va a exponer todo lo relativo al software usado e implementado a lo largo de la fase de desarrollo técnico del proyecto, con el fin de tratar de plasmar el trabajo realizado en este aspecto.

Así pues, en primer lugar se presentará la estructura software del robot, explicando todos

los detalles necesarios, y cómo han afectado en el desarrollo del proyecto. Principalmente, se tratará el uso de *ROS Control* al ser el componente más importante en la implantación de nuestros controladores en el robot.

Tras esto, se indicarán las librerías y paquetes de software no provistos por defecto con el robot que han sido necesarios para desarrollar los controladores dinámicos multiarticulares indicados a lo largo de este capítulo. Para cada uno de estos paquetes, se dará una breve descripción y se resumirá el objetivo del uso de dicho software.

Finalmente, se mostrará de forma detallada el proceso seguido para la implementación de nuestros controladores mediante ROS Control para su uso en el manipulador del TIAGo.

En la figura 3.7, se puede ver un resumen de la arquitectura software del robot.

Figura 3.7: Resumen de la arquitectura software de TIAGo. Fuente: [3]

Como se puede observar, en el robot existen dos bloques de software principales: El sistema operativo, el cual es **Ubuntu** [52] con el parche de tiempo real *Xenomai* [53], y el *middleware* para robótica, basado en *Orocos* [54] para las comunicaciones seguras y en tiempo real entre los distintos procesos ejecutándose simultáneamente.

Entrando en más detalle en lo que al *middleware* se refiere, ROS es el *middleware* estándar que se utiliza en TIAGo. Se trata de un *framework* con el objetivo de desarrollar software para robots que proporciona la funcionalidad de un sistema operativo sin serlo realmente. ROS se desarrolló originalmente en 2007 por el Laboratorio de Inteligencia Artificial de Stanford [55], y desde 2008, el desarrollo continúa principalmente en Willow Garage.

ROS proporciona al programador las características básicas asociadas a un sistema operativo como por ejemplo la abstracción del hardware, el control de dispositivos de bajo nivel, el envío de mensajes entre procesos y por supuesto el mantenimiento de paquetes. Su estructura está basada en una arquitectura de grafos donde el procesamiento toma lugar en los nodos, que pueden recibir y mandar mensajes publicados por parte de sensores, controladores, planificadores y actuadores, entre otros, como se muestra en la figura 3.8. El *middleware* está desarrollado para la distribución de Linux *Ubuntu* aunque también se está adaptando a otros sistemas operativos.

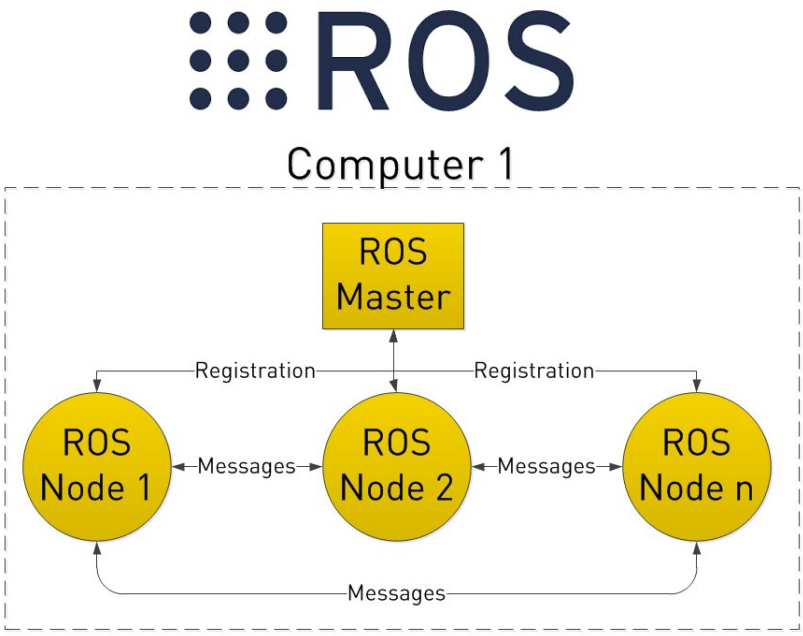


Figura 3.8: Estructura de funcionamiento de ROS. Fuente: `http://dailyautomation.sk/04-ros-robot-operating-system-zakladne-principy/`

ROS tiene dos partes básicas: El sistema operativo en sí, *ros*, descrito anteriormente y *ros-pkg*, un conjunto de paquetes aportados por su comunidad de usuarios (los llamados *stacks*) que implementan diversas funcionalidades, tales como Localización y mapeo simultáneo (SLAM), planificación, percepción, simulación, etc.

ROS es software libre bajo términos de licencia BSD. Dicha licencia otorga libertad para uso comercial e investigador, aunque las contribuciones realizadas por cada usuario en ros-pkg están bajo una gran variedad de licencias diferentes.

Así pues, el listado completo de paquetes de ROS incluidos en el robot se pueden clasificar en tres categorías distintas:

- Paquetes pertenecientes a la distribución oficial de ROS, que en el caso de TIAGo actualmente es **ROS Kinetic**.
- Paquetes específicamente desarrollados por PAL Robotics, que se incluyen en la distribución propia de la compañía, llamada *Erbium*.
- Paquetes desarrollados por el usuario

De entre los paquetes clasificados en las dos primeras categorías, a los que mayor uso se dará serán *ROS Control* y *tiago_gazebo*, que gestiona la simulación dinámica del robot en el simulador *Gazebo*. En adelante, se detallará el uso de estos dos paquetes, que permiten el diseño de controladores en ROS para su implantación en el robot, así como la simulación del TIAGo en la que poder realizar las pruebas pertinentes previamente a dar el salto al robot real.

ROS ha sido uno de los mayores avances en la industria de la robótica en los últimos años. Su desarrollo comenzó como una forma de ayudar en el desarrollo de aplicaciones robóticas, facilitando la comunicación entre sensores y algoritmos, siguiendo el paradigma de *programa una vez, prueba donde sea*

Este ha sido el patrón seguido los últimos años, y ROS se ha desenvuelto extremadamente bien en ese aspecto. Sin embargo, esta "obsesión" en la capa de más alto nivel del desarrollo de aplicaciones para robots llevó a un olvido impensable: ¿Cómo se gestionaba el control de los actuadores? ¿Cómo se calculaban las referencias de los actuadores? Del mismo modo que en el caso de las aplicaciones de alto nivel, donde el usuario final no debería preocuparse sobre el origen y destino de los datos que éste usa o produce, en el caso del control del robot tampoco debería preocuparse de qué tipo de actuadores son usados por el robot.

No obstante, a día de hoy esto ya no es un problema. **ROS Control** es el paquete que ha sido desarrollado por la comunidad de ROS [56] para permitir el acceso simple a los distintos actuadores, así como la implementación de controladores de posición, velocidad o aceleración. Usando este paquete estándar, el código del controlador es independiente del código del actuador, pudiendo uno, por ejemplo, evaluar diferentes algoritmos de control con el mismo hardware sin cambiar una línea de código, como es el caso que nos ocupa.

En la figura 3.9, se observa de forma esquemática la estructura de ROS Control, pudiendo apreciar de un vistazo sus ventajas.

Como se puede observar de la figura 3.9, la estructura de ROS Control cuenta con diversos módulos bien diferenciados. Uno de ellos es el controlador, el cual básicamente tiene comandos de entrada (consignas) y de salida (como alcanzar esa consigna), e internamente aplicará una determinada ley de control para obtener una determinada salida en función de la entrada que tenga y la realimentación del estado del robot.

Por otro lado, se tiene la interfaz hardware, la cual es una representación mediante software del robot y su hardware abstracto. La interfaz hardware almacena los recursos hardware necesarios para el controlador y envía comandos a éstos, por lo que actúa como un intermediario entre el controlador y el robot, sea real o simulado, como se mostrará más adelante.

ROS Control tiene diversas características que lo hacen realmente atractivo: Capacidad de

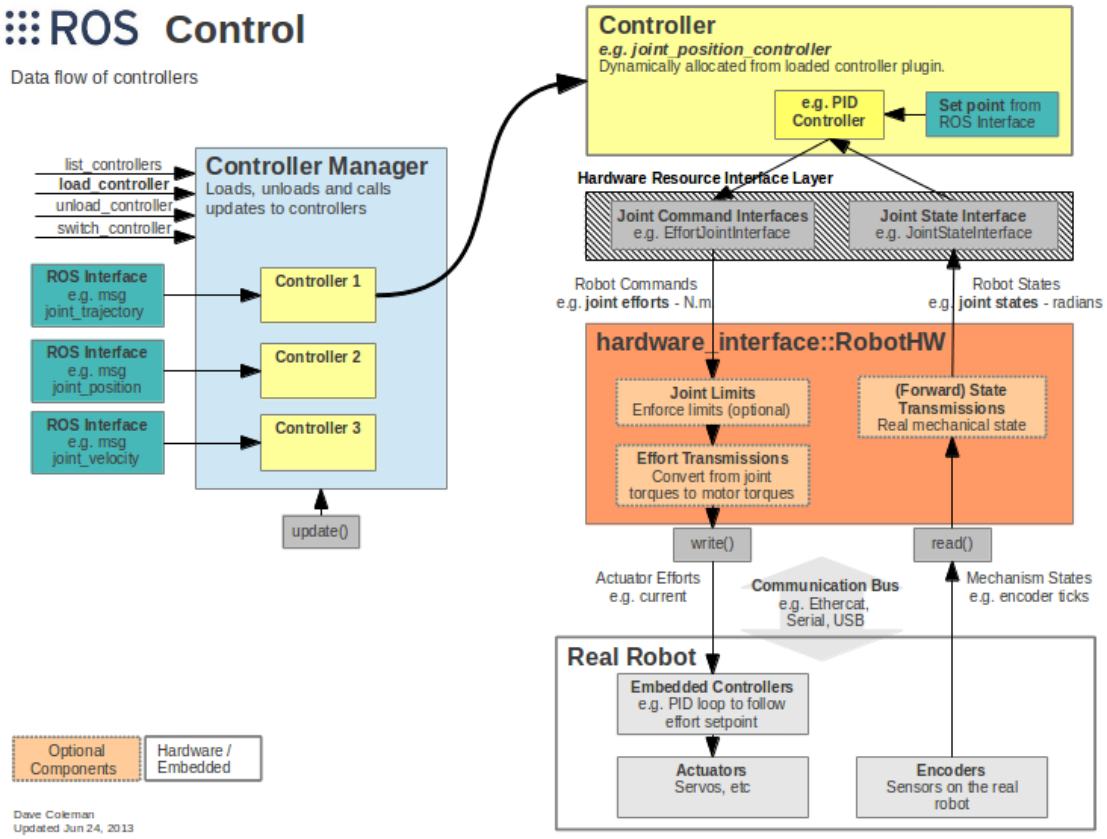


Figura 3.9: Estructura de funcionamiento de ROS Control. Fuente: `http://wiki.ros.org/ros_control`

tiempo real, que le permite ejecutar bucles de control a una frecuencia de cientos de hertz; una interfaz simple de gestión, que da acceso a los actuadores y se encarga de la resolución de conflictos; una interfaz de seguridad, que tiene constancia de las limitaciones físicas de las articulaciones y se asegura de que los comandos enviados a los actuadores estén dentro de sus límites; y un conjunto de controladores básicos listos para ser usados, aunque en este caso eso no será necesario.

Finalmente, esta separación entre el código del controlador y el hardware al que se aplicará conlleva una opción muy interesante: La simulación. Gazebo, el simulador dinámico de robots por defecto en ROS, implementa actuadores simulados compatibles con ROS Control,

por lo que se pueden evaluar los controladores implementados incluso sin disponer del robot real. Esta característica fue clave en el desarrollo del proyecto, permitiendo la verificación de los controladores implementados, y obteniendo una serie de resultados interesantes que se indicarán en el capítulo 4.

En la figura 3.10, se muestra de forma esquemática esta conexión entre Gazebo y ROS Control.

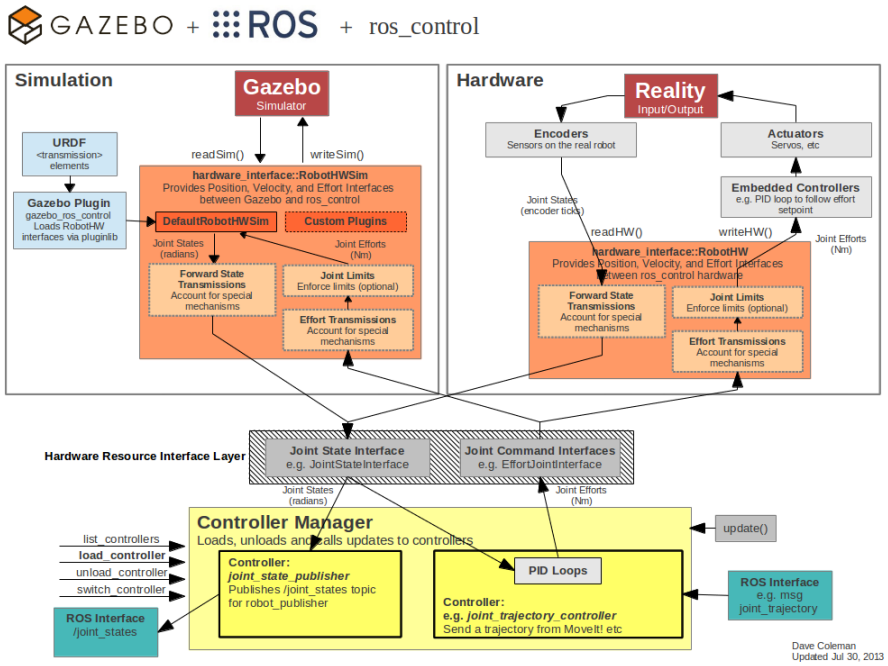


Figura 3.10: Interconexión entre Gazebo y ROS Control. Fuente: `http://gazebosim.org/tutorials/?tut=ros_control`

Con esto, a continuación se va a explicar el proceso genérico para la creación de un controlador haciendo uso de ROS Control. De forma esquemática, los pasos a seguir para su creación son los siguientes:

1. **Crear un paquete de ROS**. Crear un nuevo paquete en el espacio de trabajo que dependa de *roscpp*, *pluginlib*, *controller_interface* y *hardware_interface*.

2. **Crear el código fuente del controlador**. Se debe escribir, en lenguaje C++, el código del controlador que se desee implementar. Éste debe definir una clase derivada de la clase base *controller_interface::Controller<hardware_interface::DesiredJointInterface>*, y contener obligatoriamente los siguientes métodos:

   - ***init()***. Indica el proceso a realizar cuando se carga el controlador en el *controller_manager*.
   - ***starting()***. Indica el procedimiento a realizar cuando se arranque el controlador.
   - ***stopping()***. Indica qué hacer cuando se detenga el controlador.
   - ***updateCommand()***. Indica el proceso a llevar a cabo en cada ciclo de control. Es aquí donde se debe realizar el cálculo de la acción de control y su envío a los actuadores.

3. **Crear el *plugin* del controlador**. Se debe crear un archivo XML en el cual se indique el nombre del controlador, su tipo (posición,velocidad,esfuerzo) y la clase C++ que lo implementa, realizada en el paso anterior.

4. **Ajustar archivos *package.xml* y *CMakeLists.txt***. Modificar estos archivos para incluir en ellos el controlador recién implementado.

5. **Crear el archivo de configuración y el *launchfile* del controlador**. Crear el archivo YAML en el que se indican los recursos (articulaciones) a controlar por el robot, así como el *launchfile* que indicará los nodos a lanzar (en este caso el propio controlador) y los archivos de configuración a cargar en el servidor de parámetros de ROS.

Esta es una explicación muy superficial del proceso necesario para crear un controlador genérico en el marco de ROS y ROS Control. Si se desea una explicación pormenorizada, en [57] se dispone de un tutorial muy completo.

Así pues, con esto queda indicado cómo se puede crear un controlador en ROS Control, y al final de este apartado se expondrá en detalle el proceso realizado para implementar los

controladores dinámicos multiarticulares estudiados. Sin embargo, antes de eso es conveniente presentar la simulación dinámica del robot TIAGo en Gazebo, que nos permitirá realizar todas las pruebas que se deseen sin disponer físicamente del robot.

La simulación de robots es una herramienta imprescindible para cualquier ingeniero en robótica. Un simulador bien diseñado hace posible evaluar rápidamente algoritmos en prueba, diseñar robots, o incluso entrenar sistemas de Inteligencia Artificial (IA) en entornos realistas. **Gazebo** [58] ofrece la capacidad de simular de forma precisa y eficiente todo tipo de robots en entornos complejos, ya sean en interiores o exteriores. Para ello, cuenta con un motor de físicas robusto, gráficos de alta calidad, así como interfaces sencillas e intuitivas. Además, este simulador es gratuito, estandarizado en ROS, y cuenta con una gran comunidad detrás. En la figura 3.11, se puede observar un ejemplo de escena compleja simulada para el robot *Turtlebot*.

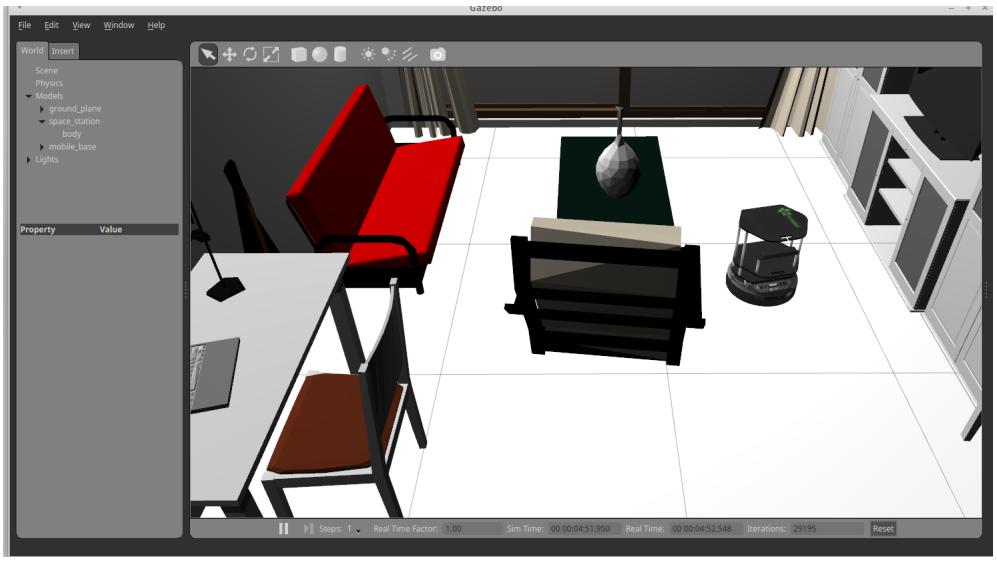


Figura 3.11: Ejemplo de escena en Gazebo. Fuente: `https://marvinferber.net/?p=128`

Como no podía ser de otro modo, existen simulaciones en diversos escenarios del robot TIAGo para Gazebo. De hecho, la simulación del robot se puede descargar públicamente desde la *Wiki* oficial del robot [59]. En esta simulación, se incluye prácticamente todo el software con el que cuenta el robot real, así como diversos escenarios en los que poder practicar con el

uso de las distintas características básicas del robot, contando con demos y tutoriales para facilitar el aprendizaje. En la figura 3.12, se ilustra una de las simulaciones disponibles para practicar en la navegación del robot.

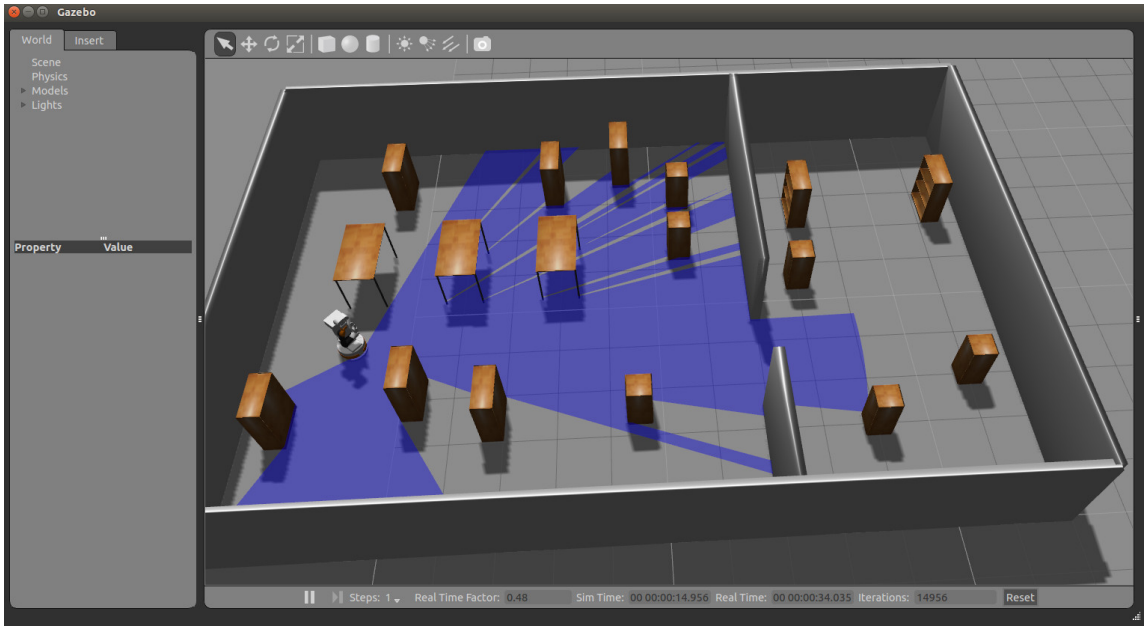


Figura 3.12: Simulación del TIAGo en interior. Fuente: `http://wiki.ros.org/Robots/TIAGo/Tutorials/Navigation/Localization`

Sin embargo, dado que en el marco de este proyecto no se necesita ningún tipo de entorno para la simulación, se hará uso de la simulación más sencilla disponible, en la cual solo se encuentra el robot TIAGo.

Así pues, es importante destacar la manera en la que Gazebo recrea el modelo del robot en la simulación. Para crear el modelo 3D del robot, el simulador necesita el archivo de descripción del robot, conocido comúnmente como archivo Unified Robot Description Format (URDF) [60].

El archivo URDF del robot es un documento XML en el cual se describe la cinemática y la dinámica del robot. Para ello, se indican las articulaciones y eslabones que forman la estructura del robot, en una relación *padre-hijo* similar a una estructura de árbol. Así pues,

para cada articulación se indica su tipo de interfaz hardware (Posición, Velocidad, Esfuerzo), así como cual es su eslabón "padre", y cual es su "hijo", como se puede ver en la figura 3.13. De esta forma, se consigue definir la cadena cinemática del robot, haciendo uso de los parámetros Denavit-Hartenberg para describir las relaciones entre las distintas articulaciones. En resumen, es el archivo donde se describe el modelo cinemático del robot.

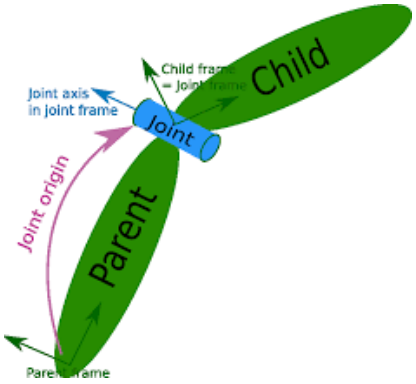


Figura 3.13: Ejemplo de funcionamiento de URDF. Fuente: `https://ni.www.techfak.uni-bielefeld.de/files/URDF-XACRO.pdf`

Además, también es posible incluir la dinámica de cada uno de los componentes del robot (masa, tensor de inercia, etc.), así como el modelo CAD de dicho componente si se dispone de él, consiguiendo así una simulación 3D mucho más visual, y un comportamiento dinámico más realista, que facilitará posteriormente el salto desde el entorno simulado al robot real.

En la figura 3.14, se puede observar de un vistazo las diversas funcionalidades del paquete URDF.

El URDF del robot es imprescindible en el caso de este proyecto, no solo por el hecho de permitir una simulación dinámica de alta fidelidad del robot, sino porque incluye de forma indirecta el modelo dinámico del brazo manipulador, cuyo control es el objetivo de la implantación de los controladores estudiados. Así pues, se necesita obtener a partir de los parámetros dinámicos de los distintos componentes del brazo manipulador del TIAGo el modelo dinámico del brazo en su conjunto.

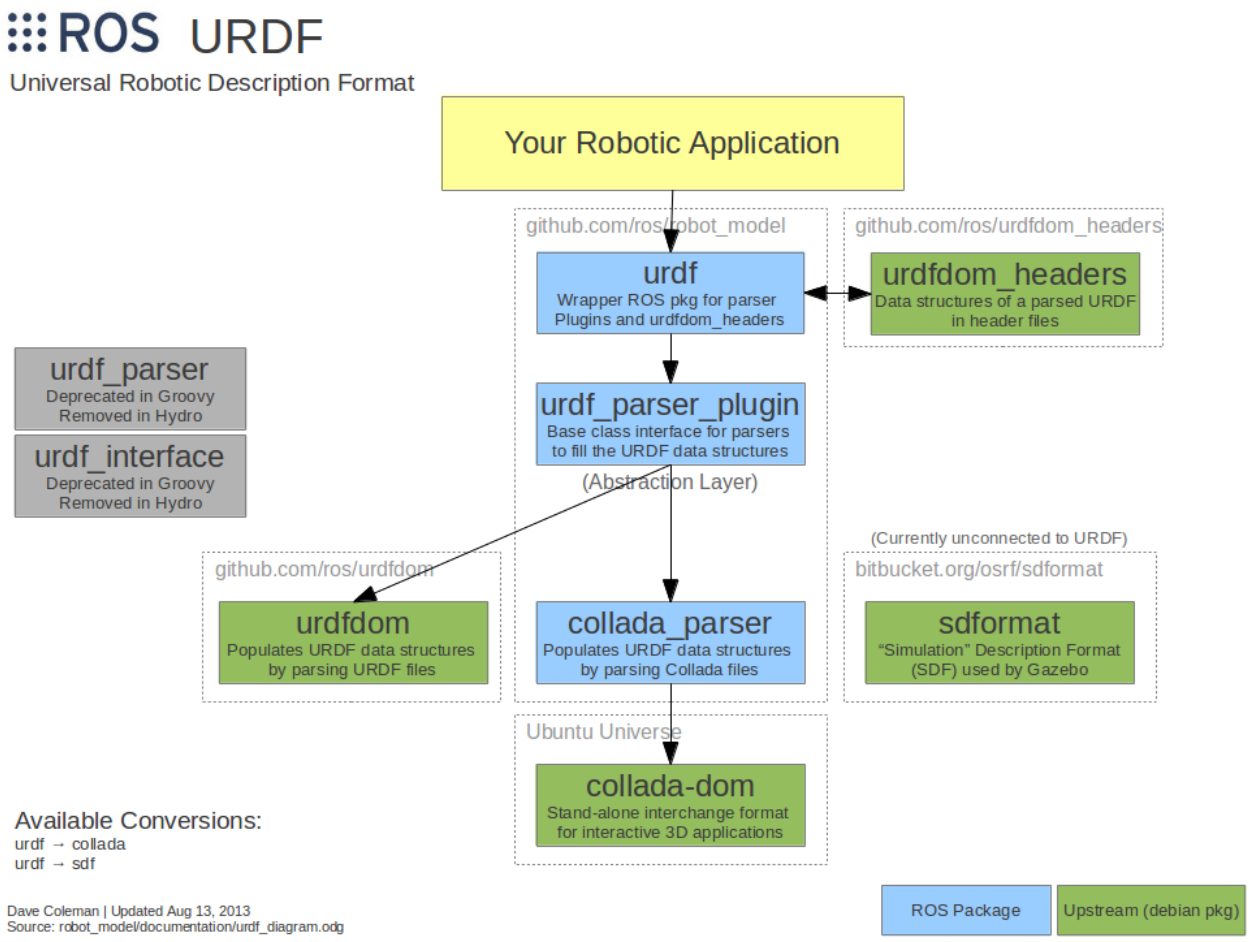


Figura 3.14: Diagrama del sistema URDF. Fuente: `http://wiki.ros.org/urdf`

Por ello, antes de explicar en detalle la implementación de los controladores, es necesario mencionar el paquete de software ajeno a los incluidos en el TIAGo que ha sido vital a la hora de conseguir el objetivo de la implantación práctica: KDL.

The Kinematics and Dynamics Library (KDL) implementa un *framework* independiente de la aplicación para el modelado y computación de cadenas cinemáticas, tales como robots, modelos humanos biomecánicos, figuras animadas por ordenador, herramientas de máquinas, etc. Este paquete provee al usuario de librerías de clases (C++) para objetos geométricos

(puntos, *frames*, líneas, ...), cadenas cinemáticas de varias familias de robots (seriales, humanoides, paralelos, móviles), así como su especificación de movimiento, interpolación, dinámica, etc.

Haciendo uso de KDL, se consiguió obtener el modelo dinámico a partir del modelo URDF del robot TIAGo. Para ello, KDL analiza dicho archivo y sus componentes, y aplica algoritmos de dinámica inversa, concretamente el algoritmo Newton-Euler [12] para obtener los valores de las matrices de inercia, Coriolis y gravedad de una determinada cadena cinemática. Además, también permite el cálculo de la matriz Jacobiana del robot, que como se ha estudiado anteriormente es imprescindible en el caso de los controladores cartesianos, ya que es el elemento que permite obtener las acciones de control a nivel articular necesarias para el control del robot manipulador.

La documentación de esta librería se puede consultar en [61], y de ese modo comprobar de primera mano todas las funcionalidades que implementa, y que son de gran interés al facilitar enormemente los cálculos a realizar por el programador.

Además, KDL hace uso a su vez de otra librería que también ha tenido un papel muy importante en la implementación de los controladores: *Eigen*.

Eigen es una librería para llevar a cabo operaciones propias del álgebra lineal en programas C++, en los cuales no existen tipos de datos como matrices o vectores en el sentido matemático. Dentro de esta librería, se implementan una gran cantidad de funcionalidades y tipos de datos, además de redefinir los operadores de suma, resta o multiplicación de todos ellos, por lo que se consigue un código limpio y elegante para programas que necesiten de gran cantidad de cálculo matricial, como son nuestros controladores dinámicos multiarticulares, tal y como se ha visto a lo largo de este capítulo. Para conocer en más detalle las funcionalidades disponibles en la librería, se puede acudir a su documentación [62].

Con esto, una vez explicado el software principal incluido en el TIAGo utilizado en este

proyecto, así como las librerias externas que han sido necesarias para llevar a cabo los cálculos propios de cada controlador, de este punto al final del capítulo se procederá a detallar la creación del paquete de software que implementa los controladores dinámicos multiarticulares estudiados.

Antes de empezar, cabe destacar que para la implementación de los controladores y su implantación en el TIAGo, se ha tomado como referencia el paquete *joint_trajectory_controller* incluido en el software creado por PAL Robotics para el robot. Este paquete implementa tanto el controlador por defecto del robot como una interfaz para la generación de trayectorias articulares mediante *splines* quínticos.

Así pues, dado que el objetivo del proyecto radica en el diseño de controladores, y no en la creación de planificadores de trayectorias, se decidió hacer uso de la interfaz de trayectorias ya implementada, la cual es de dominio público y forma parte de la simulación que se puede descargar en [59].

De vuelta a la implementación de los controladores, con el objetivo de facilitar su uso, se decidió que, en lugar de crear un paquete ROS por cada uno de los controladores, se crearía un único paquete genérico, y sería a través de un parámetro en el archivo YAML de configuración que se decidirá, ya en el código fuente, qué controlador se usará.

Una vez creado el paquete, el segundo paso como se ha indicado previamente es la implementación del código fuente del controlador, siguiendo la estructura de clase C++ con los métodos de clase necesarios. A continuación, se va a indicar el contenido de cada uno de los métodos, aunque se puede encontrar el código fuente del controlador al completo en el Anexo A.

En primer lugar, se crea el método ***init()***, en el que se deben realizar las tareas iniciales necesarias antes de poner en funcionamiento el controlador.

En este método, primeramente se leen del servidor de parámetros de ROS los nombres de las articulaciones a manejar, para así posteriormente obtener sus *manejadores* y de ese modo poder leer información de cada articulación y enviar las acciones de control. Tras esto, se leen del servidor de parámetros los valores iniciales de las matrices de ganancias PD, así como el tipo de controlador dinámico que se ha escogido para el robot en ese caso (PD con prealimentación, PD+, Par-Calculado, Cartesiano basado en Par-Calculado u Óptimo cartesiano). A modo de ejemplo, en el código 3.1 se ilustra cómo leer un parámetro del servidor de ROS.

Código 3.1: Lectura de parámetros del servidor de ROS

```
1
2 // Get joint names from parameter server
3     if (!controller_nh_ptr_->getParam("joints", joint_names_)){
4       ROS_ERROR("Could not load joint names from parameter server");
5     }
6
7     // Obtain proportional and derivative gains stated at the YAML config file
8     double kp_default, kv_default = 0.0;
9
10    if (!controller_nh_ptr_->getParam("Kp", kp_default)){
11      ROS_ERROR("Could not load proportional gain. Default value Kp=500");
12    }
13
14    if (!controller_nh_ptr_->getParam("Kv", kv_default)){
15      ROS_ERROR("Could not load derivative gain. Default value Kv=50");
16    }
17
18     // Obtain required controller type. Default: PD+
19    if (!controller_nh_ptr_->getParam("controller_type", controller_type)){
20      ROS_ERROR("Could not load controller type. Default: PD+");
21    }
```

Tras esto, se realiza un paso bastante interesante, consistente en el hecho de usar la librería *dynamic_reconfigure* [63] para indicar que los valores de las ganancias proporcional y derivativa de cada articulación se puedan modificar en tiempo de ejecución, evitando así tener que parar y relanzar el controlador constantemente durante el proceso de sintonía. Este proceso es ligeramente largo, pero se puede consultar en el Anexo A si se tiene interés.

A continuación, de nuevo se accede al servidor de parámetros para leer los datos relativos a la relación entre corriente y par articular de los motores del brazo, ya que como se dijo en el apartado de hardware, en el robot real no se puede realizar un control directo al uso, sino un control en corriente.

Por último, se lleva a cabo un paso muy importante, como es la obtención de la cadena cinemática del brazo con KDL. Para esto, en primer lugar se lee el modelo URDF del robot al completo del servidor de parámetros, y tras esto se convierte en un objeto "árbol" de KDL. Sin embargo, para este proyecto solo se requiere el modelo del brazo manipulador del TIAGo, por lo que se indica que solo se quiere obtener la cadena cinemática correspondiente a las articulaciones y eslabones del brazo. Esto se realiza de forma muy sencilla tal y como se muestra en el código 3.2.

Código 3.2: Obtención de la cadena cinemática del manipulador

```
// Obtain KDL Tree
  urdf::Model tiago_model;
  std::string paramName = "/robot_description";
  if (!tiago_model.initParam(paramName)){
    ROS_ERROR("Failed to parse urdf robot model");
    return false;
  }

  if (!kdl_parser::treeFromUrdfModel(tiago_model, tiago_kdl)){
    ROS_ERROR("Failed to construct kdl tree");
    return false;
  }

  // Obtain kinematic chain corresponding to TIAGo arm
  tiago_kdl.getChain("torso_lift_link","arm_7_link",chain);
```

Cabe destacar que, aunque solo se vayan a controlar las 4 primeras articulaciones, propias al brazo del manipulador únicamente, en la cadena cinemática se consideran los 7 GDL del manipulador, con el propósito de tener en cuenta todos los elementos de éste en el cálculo de la dinámica, obteniendo así resultados fieles a la realidad.

Con esto, queda resumido todo el proceso a realizar en la inicialización del controlador,

el cual básicamente consiste en la obtención de parámetros del servidor correspondiente de ROS, y en la obtención de la cadena cinemática del manipulado del TIAGo leyendo el URDF con KDL.

Seguidamente, en la clase que implementa nuestros controladores se tienen los métodos ***starting()*** y ***stopping()***, en los cuales únicamente se manda un comando de par articular nulo. En el caso de ***starting()***, esto se hace para que el bucle de control se lance cuando no se esté ejerciendo ningún par a las articulaciones, mientras que en ***stopping()*** se anulan los pares por motivos de seguridad, con el objetivo de que ante una parada imprevista del controlador, se dejen de enviar pares al brazo, ya que de lo contrario éste se seguiría moviendo con los pares indicados por la última acción de control.

De este modo, se llega al método ***updateCommand***, en el cual se implementa todo lo relativo al controlador propiamente dicho, ya que lo que se describa en este método será ejecutado en cada ciclo de control.

Así, en el bucle de control el primer paso es la realimentación del estado del robot, para lo cual se consultan el estado actual de posición, velocidad y par articular del robot, así como las posiciones, velocidades y aceleraciones articulares deseadas, y por último el error en posición y velocidad.

Tras esto, se lleva a cabo una instrucción condicional *If-Else if-Else* en la cual, según el controlador que se haya escogido en el archivo YAML de configuración, se realizarán los cálculos necesarios para aplicar la ley de control deseada. Por ejemplo, en el caso de que se haya escogido un control PD con prealimentación, el fragmento de código ejecutado sería el mostrado en el código 3.3.

Código 3.3: Código del controlador PD con prealimentación

```
1
2    else if(controller_type.compare("PD_Feedforward")==0)
3    {
4      dynamics.JntToGravity(desired.q,G);
```

```
    dynamics.JntToCoriolis(desired.q,desired.qdot,C);
    dynamics.JntToMass(desired.q,M);

    // Obtain torque needed for each joint
    tau.data = Kp*error.q.data + Kv*error.qdot.data + M.data*desired.qdotdot.data + C.data + G.data - F * current.qdot.data;
    for (unsigned int i = 0; i < n_joints; ++i)
    {
      // Effort command sending
      const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
      if(!std::isnan(command))
        (*joint_handles_ptr_)[i].setCommand(command);
    }

  }
```

Cabe indicar que, puesto que se ha hecho uso de la interfaz para la generación de trayectorias articulares incluida por defecto en el robot, a la hora de implementar los controladores cartesianos se necesitó convertir en primer lugar la referencia articular en una referencia cartesiana, para lo cual se hizo uso de nuevo de KDL. Tras esto, se calculó el error en espacio cartesiano y se obtuvo la matriz Jacobiana del robot, así como otros valores necesarios como su derivada temporal o su pseudo-inversa.

Esta adaptación, pese a no ser lo ideal, no afecta en el análisis de los resultados obtenidos por cada controlador, puesto que a ojos de éste la referencia con la que trabaja es cartesiana, independientemente de las conversiones que hubiesen sido necesarias previamente.

Finalmente, una vez enviada la acción de control obtenida usando la ley de control escogida, se publican en dos *topics* de ROS los valores del error en posición (articular o cartesiana, dependiendo del controlador) y del par articular que cada una de las cuatro primeras articulaciones ejerce durante el seguimiento de la trayectoria. Esto se implementó con el objetivo de almacenar los datos de cada prueba realizada en un fichero de texto, para así poder disponer posteriormente de una gran cantidad de datos para su análisis y evaluación.

De este modo, queda resumido el código fuente implementado dentro del paquete desarro-

llado, aunque de nuevo se recuerda que en el Anexo A se dispone de éste en su totalidad si se desea realizar cualquier consulta.

El siguiente paso es crear el *plugin* del controlador, que será el que defina el código fuente anterior como un nuevo controlador dentro de la estructura de ROS Control. Este *plugin*, escrito en XML, toma una forma muy sencilla, mostrada en el código 3.4.

Código 3.4: *Plugin* del controlador implementado

```
1
2 <library path="lib/libdirect_dynamic_controller">
3     <class name="effort_controllers/DirectDynamicController" type="effort_controllers::↩
          ↪ DirectDynamicController" base_class_type="controller_interface::ControllerBase">
4         <description>
5             The DirectDynamicController executes joint−space trajectories on a set of joints. This variant ↩
                  ↪ obtains directly the effort that is needed at each joint taking into account the robotic arm↩
                  ↪  dynamics (Inertia, Coriolis and Gravity), which improves performance.
6         </description>
7     </class>
8 </library>
```

Como se puede observar del código 3.4, tan solo se nombra el tipo del controlador, para poder usarlo después en un archivo de configuración, y adicionalmente se da una pequeña explicación del objetivo del controlador.

Así pues, con el *plugin* ya creado, a continuación es necesario modificar los archivos *package.xml* y *CMakeLists.txt* estándar de los paquetes ROS, con el fin de indicar, en el primer archivo, las dependencias software añadidas durante la implementación del controlador, mientras que en el segundo además de esto se debe denotar la existencia de un nuevo código fuente que debe ser compilado. Este paso, sin embargo, no es exclusivo de la creación de controladores en ROS Control, sino que es la práctica habitual para cualquier paquete creado dentro de ROS, por lo que no se va a profundizar más en este aspecto.

Como último paso antes de poder usar el controlador implementado, se deben crear el archivo YAML de configuración y el *launchfile* que se ejecutará cuando se desee cargar el controlador en el *controller manager*.

En el fichero YAML de configuración tan solo se necesita indicar el nombre del controlador que se quiere cargar en el servidor, su clase (que será el indicado en el *plugin*), las articulaciones que va a controlar, el valor por defecto de las ganancias PD, y por último el tipo de controlador dinámico que se desea de entre los implementados. Un ejemplo de fichero de este tipo es el mostrado en el código 3.5.

Código 3.5: Archivo YAML de configuración del controlador

```
1
2 arm_dynamic_controller:
3   type: "effort_controllers/DirectDynamicController"
4   joints:
5     - arm_1_joint
6     - arm_2_joint
7     - arm_3_joint
8     - arm_4_joint
9
10  Kp: 100.0
11  Kv: 20.0
12  controller_type: "PD+"
```

Con un archivo de configuración de las características del código 3.5, se está indicando que se va a cargar en el *controller manager* un nuevo controlador de nombre "arm_dynamic_controller", el cual es un controlador del tipo *effort_controllers/DirectDynamicController* (El definido en el código 3.4), que va a controlar las 4 primeras articulaciones del robot, cuyas ganancias PD iniciales serán $K_p = 100\boldsymbol{I}$ y $K_v = 20\boldsymbol{I}$, y que se va a utilizar el controlador PD+ de entre los controladores dinámicos disponibles en el paquete. Además, este archivo no requiere de compilación, por lo que entre ejecuciones se puede cambiar el controlador dinámico escogido sin necesidad de recompilar el paquete modificando exclusivamente el parámetro correspondiente de este archivo.

Finalmente, el *launchfile* del controlador es el archivo que, al ser ejecutado mediante la instrucción *roslaunch*, indicará los archivos de configuración a cargar en el servidor de parámetros de ROS y cargará el propio controlador dentro del servidor del *controller manager*. En el código 3.6, se expone el archivo implementado para este controlador.

Código 3.6: *Launchfile* del controlador implementado

```
<launch>
  <rosparam command="load" file="$(find direct_dynamic_controller)/config/
      ↪ direct_dynamic_controller.yaml" />

  <!-- Set up controller -->
  <arg name="controller_list"
       value="arm_dynamic_controller"/>

  <arg name="simulation" default="false"/>

  <!-- Load the motor parameters -->
  <group if="$(arg simulation)">
    <rosparam command="load" file="$(find direct_dynamic_controller)/config/
        ↪ motor_params_simulation.yaml" />
  </group>
  <group unless="$(arg simulation)">
    <rosparam command="load" file="$(find direct_dynamic_controller)/config/motor_params.yaml" />
  </group>

</launch>
```

Se puede observar del código 3.6 como en primer lugar se carga el archivo YAML de configuración del código 3.5, después se incluye en la lista de controladores el controlador indicado en dicho archivo de configuración, y finalmente, en función de si se trata o no de una simulación, se cargan unos parámetros de los motores de las articulaciones u otros. Esto último es por el hecho de que en simulación sí se lleva a cabo un control directo, mientras que en el robot real se controla la corriente enviada a los motores, que se puede traducir a par articular multiplicando por una relación dependiente de los parámetros del motor.

Con esto, quedaría descrito el proceso realizado para la creación del paquete de ROS Control en el que se implementan los controladores dinámicos multiarticulares estudiados en este capítulo. Para poder emplearlos en el robot TIAGo, basta con copiar el paquete creado dentro del espacio de trabajo ROS del robot, recompilarlo, y ya se estaría en disposición de evaluarlos y analizar los resultados obtenidos, lo cual se va a llevar a cabo en el capítulo 4.

4. Resultados

En este cuarto capítulo del proyecto, se van a exponer los resultados obtenidos tras la experimentación realizada haciendo uso de los controladores dinámicos multiarticulares, cuya justificación matemática e implementados se ha mostrado en el capítulo 3.

Así pues, en primer lugar se van a mostrar los resultados obtenidos en la simulación dinámica del robot TIAGo en Gazebo. Para esto, en primer lugar se expondrán los resultados obtenidos con los controladores articulares *PD con prealimentación*, *PD+* y *Par-Calculado* en entorno simulado, observando el comportamiento de cada uno de ellos en condiciones ideales. Tras esto, se procederá a la evaluación de los controladores cartesianos, tratando de establecer en este caso una diferenciación entre el control cartesiano basado en Par-Calculado y el control óptimo.

A continuación, se mostrarán los resultados recabados del robot TIAGo real en las instalaciones de PAL Robotics. En este caso, se además de repetir el proceso que se hará para la evaluación de los controladores en la simulación, se indicarán también las diferencias más importantes entre el entorno simulado y el robot real, que como se verá más adelante condicionan en gran medida los resultados obtenidos. Además, se mostrará de forma breve el desarrollo de un control visual basado en posición llevado a cabo *in situ* en las instalaciones de la empresa.

Finalmente, se llevará a cabo un análisis general de los resultados obtenidos, así como una comparativa de éstos tanto entre el entorno simulado y el robot real, como entre los propios controladores, tratando de justificar razonadamente el por qué de dichas diferencias.

4.1. Simulación dinámica en Gazebo

En esta sección, se va a mostrar un extracto de todas las pruebas que se han realizado en simulación para cada uno de los controladores dinámicos multiarticulares implementados para el robot TIAGo en ROS Control. Para ello, en primer lugar se indicará el entorno simulado del que se ha hecho uso, así como la trayectoria que se ha decidido emplear como referencia para la evaluación de todos los controladores. Tras esto, se mostrarán los resultados obtenidos por cada controlador por separado en distintas pruebas con diversos valores de las ganancias K_p y K_v, observando así su comportamiento.

Como se indicó en la sección dedicada al apartado software en el capítulo 3, dentro de la arquitectura software de TIAGo se encuentra el paquete *tiago_gazebo* que incluye la simulación del robot en diversos entornos, como por ejemplo el mostrado en la figura 3.12. Sin embargo, para este proyecto tan solo se requiere contar con el robot en la simulación, puesto que el objetivo es el de analizar los controladores dinámicos multiarticulares para el seguimiento de una trayectoria que han sido implementados.

Con esto en mente, se escogió la simulación más simple de la que dispone el robot, en la cual se encuentra éste únicamente, como se observa en la figura 4.1.

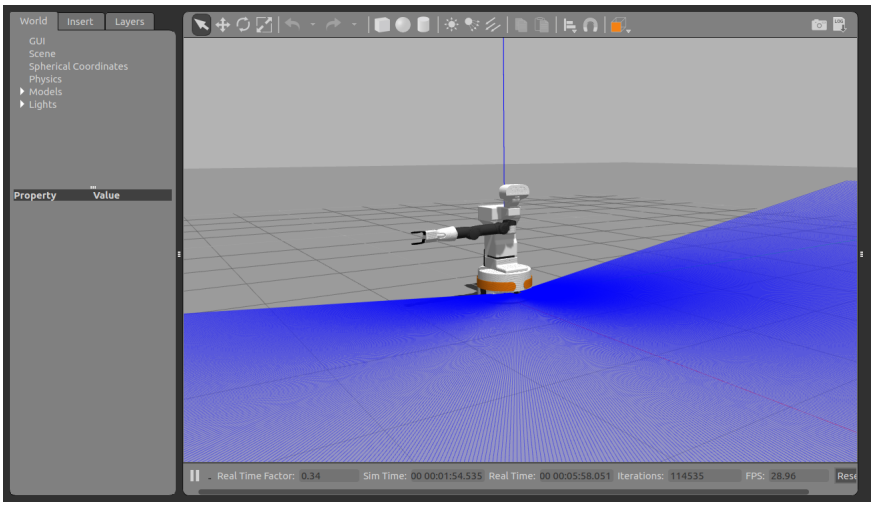
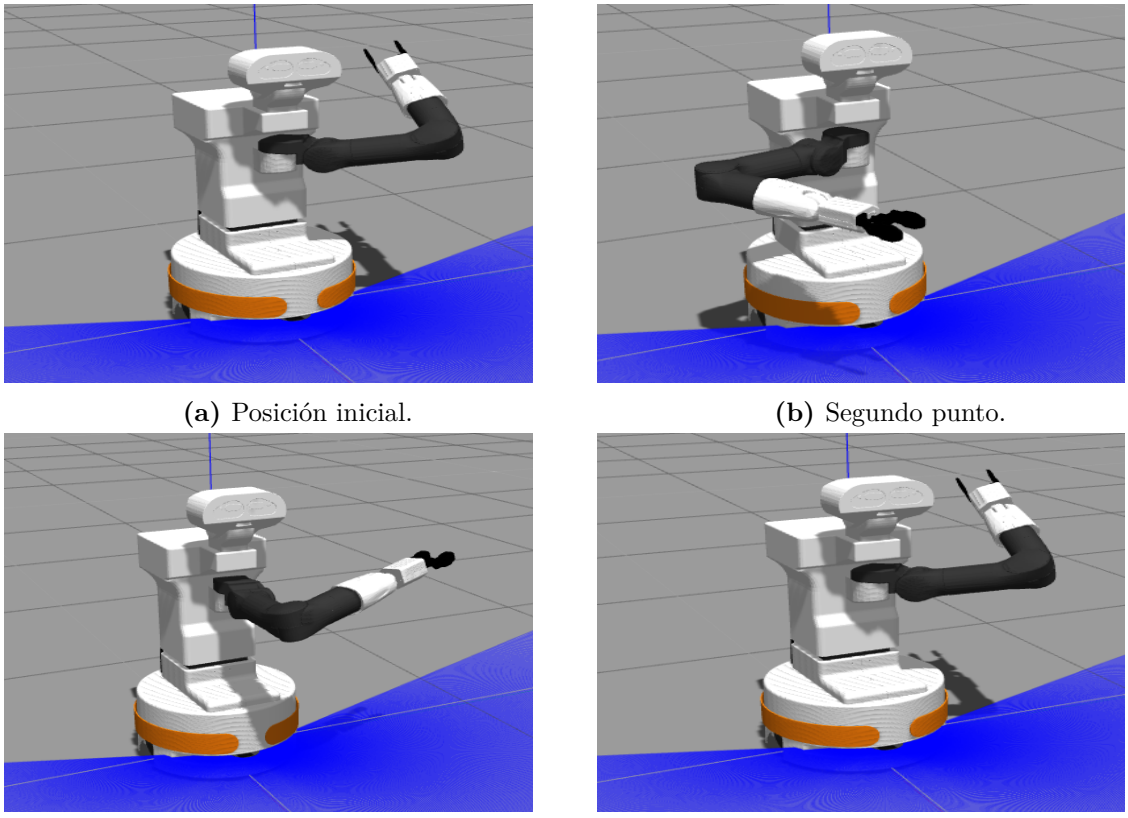


Figura 4.1: Simulación Gazebo del robot TIAGo.

En este entorno, se tuvo que escoger una trayectoria de referencia para todas las pruebas, con el fin de obtener resultados que se pudiésen comparar entre ellos. Con este objetivo en mente, se trató de que la trayectoria empleada implicase un movimiento suficiente de las articulaciones del robot, pudiendo así comprobar el desempeño de los controladores correctamente. En la figura 4.2 se muestran las etapas por las que pasa el brazo del robot en dicha trayectoria:



(a) Posición inicial.

(b) Segundo punto.

(c) Punto intermedio entre 4.2b y 4.2a.

(d) Punto final de trayectoria.

Figura 4.2: Trayectoria empleada para la evaluación de los controladores

Se puede observar de la trayectoria ilustrada en la figura 4.2 como el brazo del robot se mueve por toda la parte frontal del espacio de trabajo, implicando todas las articulaciones de éste en el movimiento. Además, cabe indicar que toda la trayectoria se lleva a cabo en un periodo de 6 segundos, y a una velocidad de $0.5\frac{rad}{s}$. El código que invoca el planificador de trayectorias para generar éste movimiento se puede consultar en el Anexo B.

Así pues, en el próximo apartado se van a examinar los resultados obtenidos con los controladores articulares implementados.

4.1.1. Control dinámico articular

En este apartado, se van a evaluar los resultados obtenidos con los distintos controladores articulares implementados en este proyecto. Para cada uno de ellos, se expondrán diferentes casos de experimentación, y se mostrará el comportamiento obtenido en cada caso, ilustrado por dos gráficas, una de error en posición articular, y otra de par articular ejercido en cada momento de la trayectoria.

Con esto, el siguiente paso ya podría ser la evaluación de los controladores dinámicos. Sin embargo, es conveniente obtener también cuál es el comportamiento del controlador que el robot incorpora por defecto, con el fin de comparar su desempeño con el de los estudiados en este proyecto.

El controlador por defecto del brazo de TIAGo es un controlador PID en posición de alto nivel, el cual recibe como consigna la posición deseada en cada momento de la trayectoria, introduce dicha referencia en un bucle de control PID, y envía como acción de control una posición, que posteriormente la interfaz hardware de ROS Control se encargará de convertir en un valor de par articular a enviar a los actuadores. Este tipo de controladores tienen diversos problemas, como por ejemplo su comportamiento a altas velocidades, o su reacción a perturbaciones o choques con el entorno, dejando el manipulador totalmente rígido, lo cual no es lo más deseable en un robot de clara orientación colaborativa.

Así pues, para la trayectoria mostrada en la figura 4.2, los resultados obtenidos con el controlador estándar del robot son los mostrados en la figura 4.3.

Como se puede observar de la figura 4.3, en lo que a valores absolutos propiamente dichos respecta, el controlador por defecto del TIAGo obtiene unos resultados bastante correctos, con errores bajos en cada articulación, y no ejerce un par articular excesivo en ningún mo-

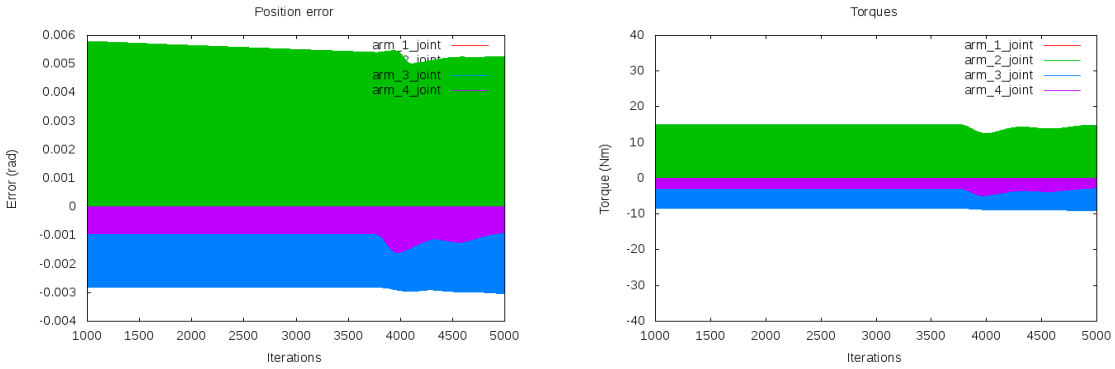


Figura 4.3: Resultados del controlador en posición por defecto.

mento, lejos de alcanzar los máximos admisibles por los motores, especificados en la Tabla 3.1.

Sin embargo, también se aprecia un comportamiento extremadamente oscilante del sistema, lo cual nunca es deseable por las vibraciones internas que provoca, que a la larga pueden causar daños al manipulador.

Así pues, en adelante se comprobará si el desempeño de los distintos controladores dinámicos mejora los resultados obtenidos con el controlador por defecto.

4.1.1.1. Control PD con prealimentación

En esta sección se abordará la evaluación de los resultados obtenidos con el controlador PD con prealimentación, cuya ley de control viene dada por la ecuación (3.3).

Así pues, se decidió empezar la fase de experimentación de este controlador con ganancias bajas, para así ir aumentándolas en las distintas pruebas y observando como varía el comportamiento del robot. En la figura 4.4 se pueden ver los resultados de la primera prueba con este controlador.

Se puede apreciar en la figura 4.4 como para este controlador desaparece completamente la oscilación extrema que se tenía en el controlador por defecto, y cómo los pares articulares siguen lejos de saturar. Sin embargo, en este caso el valor absoluto del error en posición es

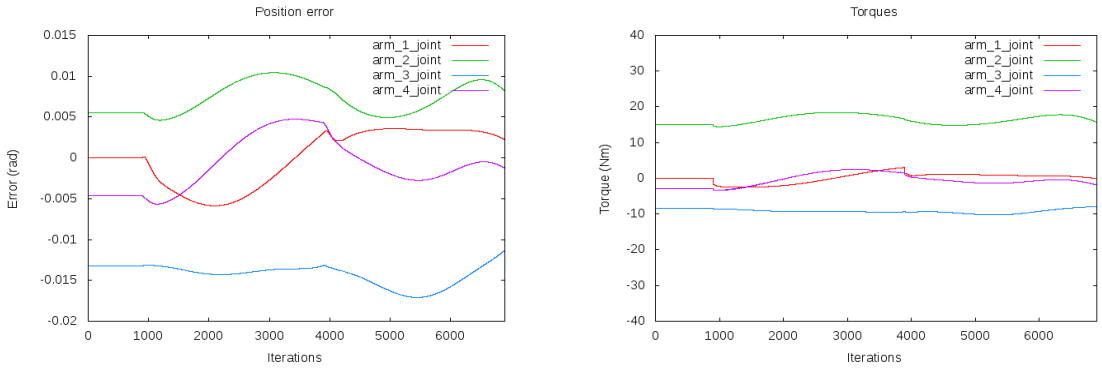


Figura 4.4: Control PD con prealimentación. $K_p = 250\boldsymbol{I}, K_v = 25\boldsymbol{I}$

mayor que en el controlador estándar, por lo que se va a proceder al ajuste de las ganancias PD del controlador. En las figuras 4.5 a 4.9, se ilustran los resultados obtenidos conforme se han ido aumentando dichas ganancias.

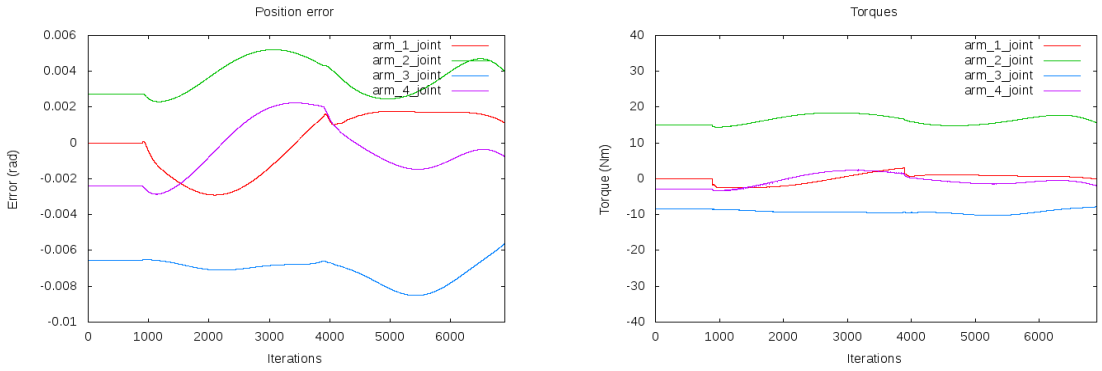


Figura 4.5: Control PD con prealimentación. $K_p = 500\boldsymbol{I}, K_v = 50\boldsymbol{I}$

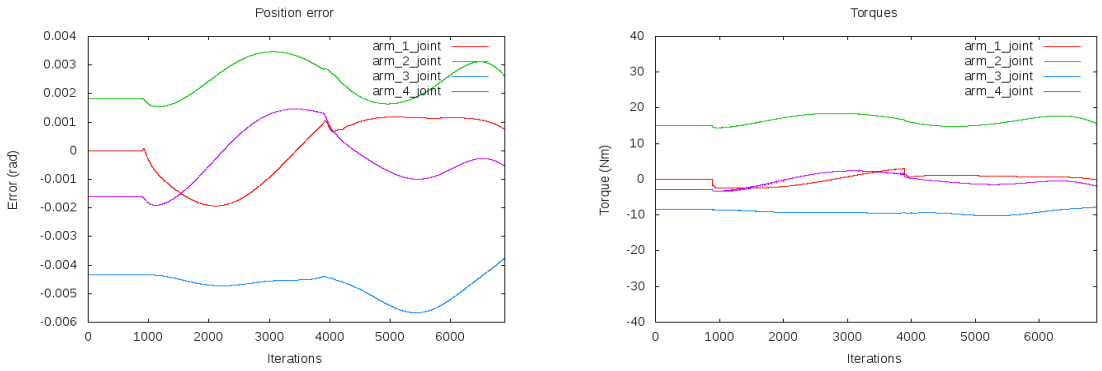


Figura 4.6: Control PD con prealimentación. $K_p = 750\boldsymbol{I}, K_v = 75\boldsymbol{I}$

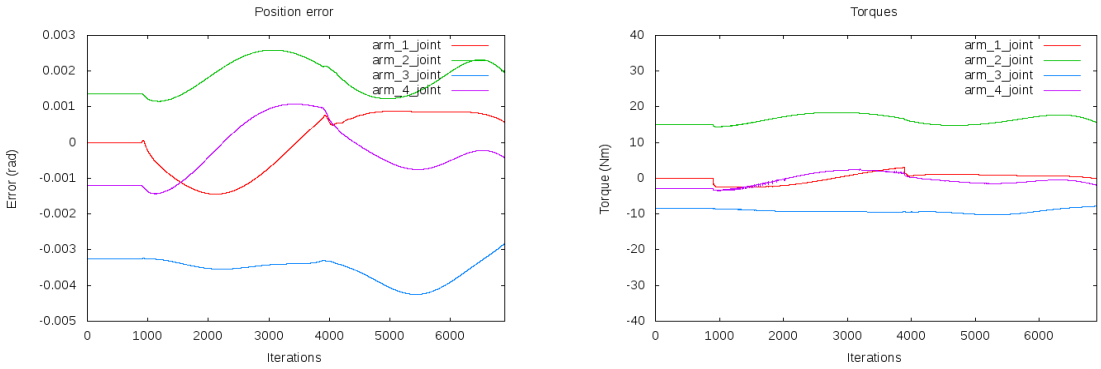


Figura 4.7: Control PD con prealimentación. $K_p = 1000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

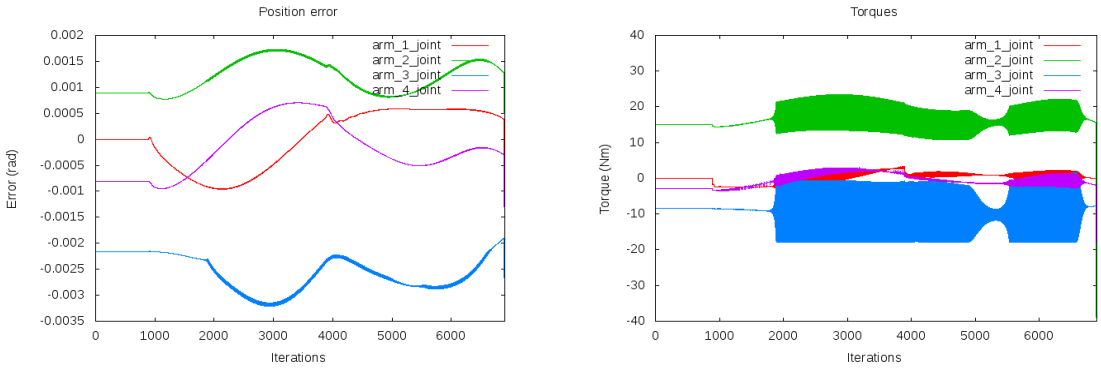


Figura 4.8: Control PD con prealimentación. $K_p = 1500\boldsymbol{I}, K_v = 150\boldsymbol{I}$

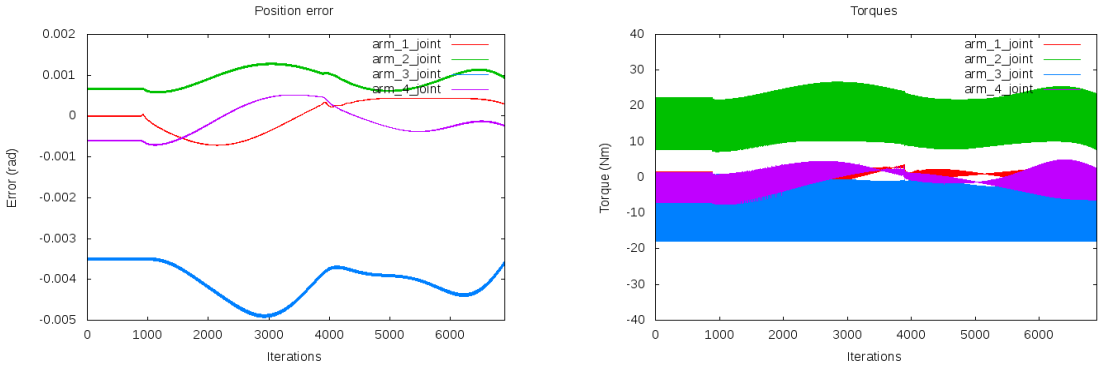


Figura 4.9: Control PD con prealimentación. $K_p = 2000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

De las figuras anteriores se puede observar que conforme vamos aumentando las ganancias K_p y K_v del controlador, éste va reduciendo su error en posición progresivamente. Sin em-

bargo, al llegar a la fígura 4.8, los valores de las ganancias empiezan a ser demasiado altos y aparecen las primeras oscilaciones en el par articular, que se vuelven mucho mayores si se incrementan aún más los valores de las ganancias, como se ilustra en la figura 4.9, donde este hecho lleva incluso a aumentar el error en posición.

Otro dato curioso es el hecho de que las articulaciones 2 y 3 son las que más par ejercen, así como las que más error acumulan. Esto se debe muy probablemente a que son las dos articulaciones sobre las que más afecta la fuerza de la gravedad al soportar la mayor parte del peso del manipulador extendido, como se puede inferir de la figura 3.6. No sucede así con la primera articulación debido a su estructura, ya que el eje de giro coincide con la dirección del vector de gravedad, y por tanto no se genera par en dicha articulación.

Así pues, se observa también que el mejor caso de entre los expuestos en las figuras 4.4 a 4.9 se da en el caso de la figura 4.7, caso en el que se emplean las ganancias $K_p = 1000\boldsymbol{I}$ y $K_v = 100\boldsymbol{I}$. Para esta configuración del controlador, se consigue que el error máximo en la trayectoria sea de $0.004rad$ (0.23) en la articulación 3, menor que en el controlador por defecto, y en el caso del error medio no supera los $0.002rad$ para el resto articulaciones, además de la clara ventaja que supone eliminar las oscilaciones.

Por tanto, con este controlador se ha logrado mejorar el desempeño del manipulador para la trayectoria evaluada, con lo que validamos el primero de los controladores diseñados.

4.1.1.2. Control PD+

En este apartado se expondrán los resultados obtenidos con el controlador PD+, cuya ley de control viene dada por la ecuación (3.8).

Tal y como se trató en el capítulo 3, la mayor diferencia a nivel teórico entre este controlador y el PD con prealimentación era la facilidad del ajuste de las ganancias. Sin embargo, a nivel práctico ambos controladores son prácticamente idénticos, cambiando únicamente que en el caso del PD con prealimentación se evaluaba el modelo dinámico (2.1) con la posición y

velocidad deseadas, $\boldsymbol{q}_d$ y $\dot{\boldsymbol{q}}_d$, mientras que para el control PD+ se obtiene el modelo en base a la posición y velocidad actuales del robot, $\boldsymbol{q}$ y $\dot{\boldsymbol{q}}$.

Así pues, al igual que en el caso anterior, se optó por empezar la fase de experimentación de este controlador con ganancias bajas, aumentándolas progresivamente en las distintas pruebas y observando como varía el comportamiento del robot. En la figura 4.10 se pueden ver los resultados de la primera prueba con el control PD+.

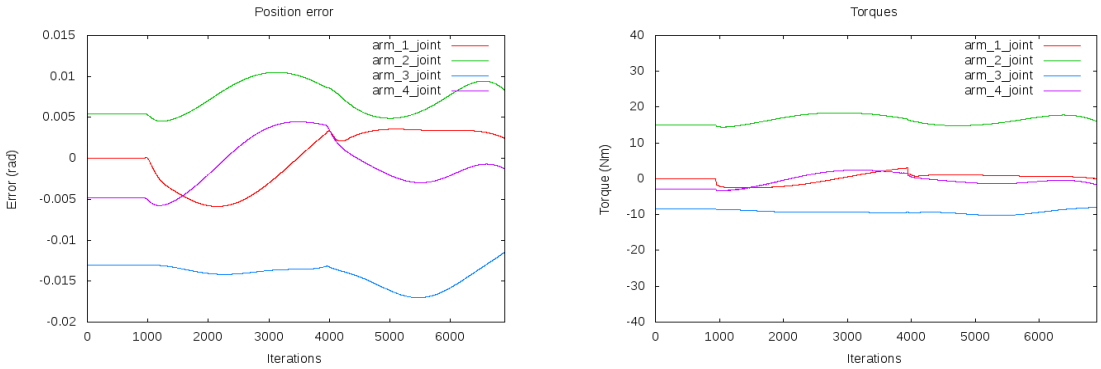


Figura 4.10: Control PD+. $K_p = 250\boldsymbol{I}, K_v = 25\boldsymbol{I}$

Tal y como se observa en la figura 4.10, el resultado inicial obtenido es muy similar al del caso del control PD con prealimentación, mostrado en la figura 4.4. Este hecho no es una sorpresa, puesto que estamos trabajando en un entorno simulado y no existen perturbaciones no modeladas, que son las que el control PD+ trata mejor al obtener el modelo dinámico del estado actual, y no del estado deseado como en el PD con prealimentación. Por tanto, dada la similitud entre ambos controladores, todo hace indicar que los resultados obtenidos serán muy parecidos. En las figuras 4.11 a 4.15, se ilustran los resultados obtenidos conforme se ha ido aumentando el valor de las ganancias PD.

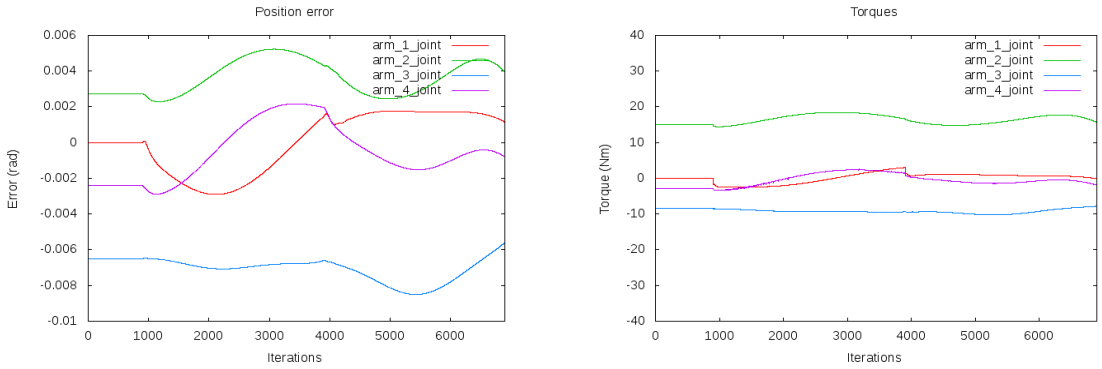


Figura 4.11: Control PD+. $K_p = 500\boldsymbol{I}, K_v = 50\boldsymbol{I}$

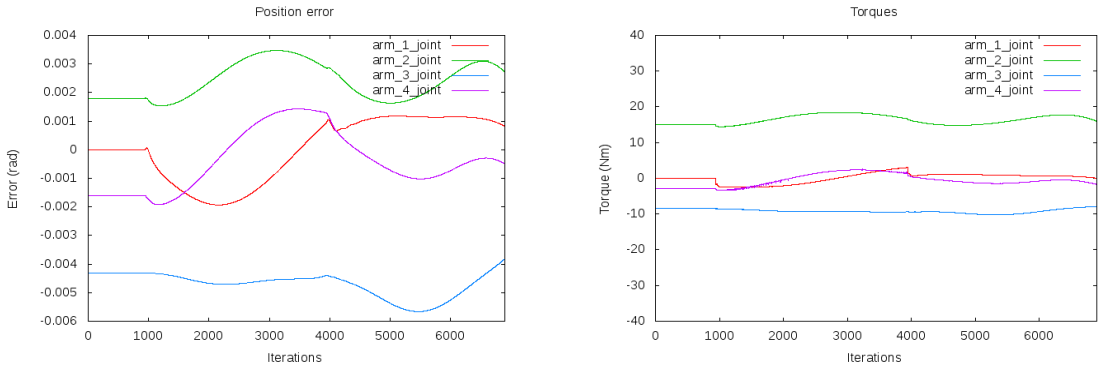


Figura 4.12: Control PD+. $K_p = 750\boldsymbol{I}, K_v = 75\boldsymbol{I}$

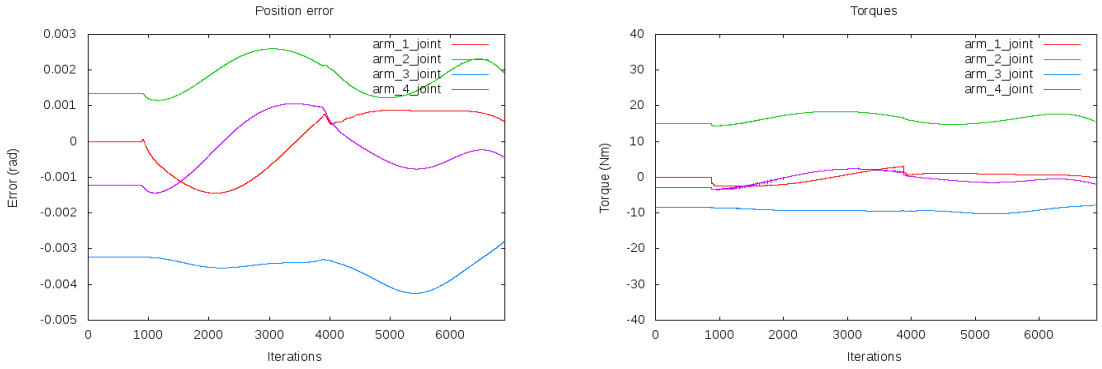


Figura 4.13: Control PD+. $K_p = 1000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

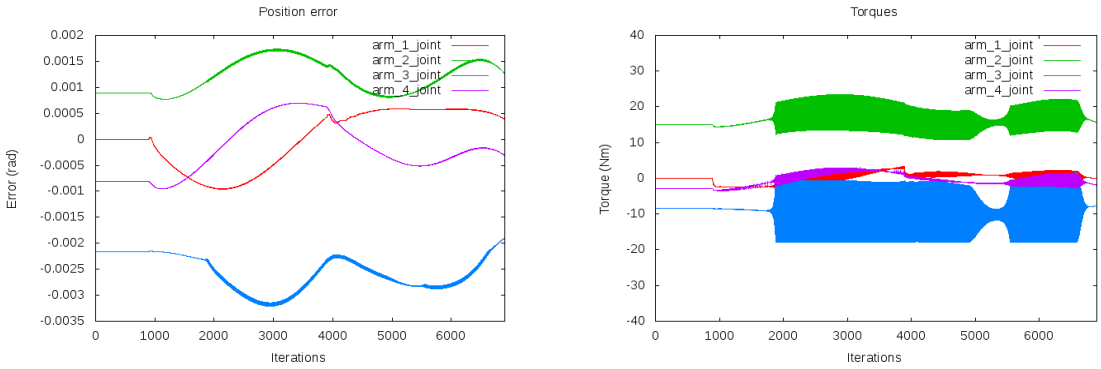


Figura 4.14: Control PD+. $K_p = 1500\boldsymbol{I}, K_v = 150\boldsymbol{I}$

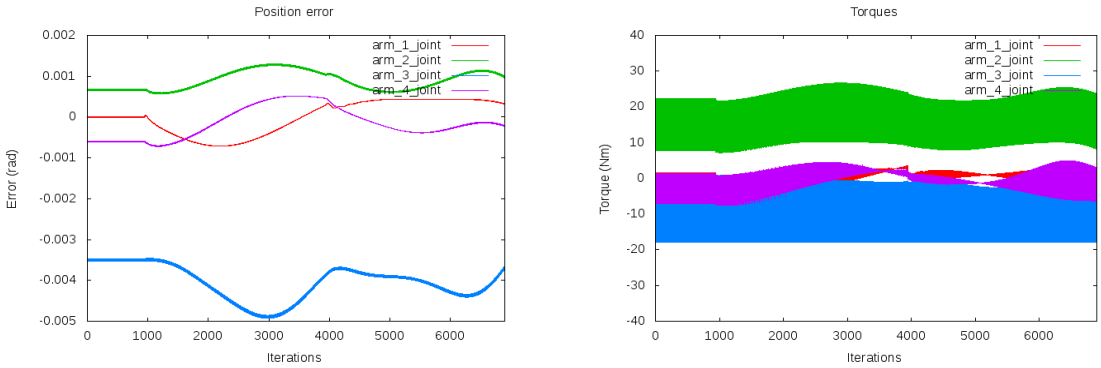


Figura 4.15: Control PD+. $K_p = 2000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

Como se observa en las figuras anteriores, el comportamiento de este controlador es prácticamente idéntico en todos los casos al PD con prealimentación. De nuevo, el error en posición va disminuyendo conforme se incrementa el valor de las ganancias PD, hasta llegar al punto en el que empieza a oscilar en la figura 4.14, terminando de empeorar si aumentamos ligeramente las ganancias como en el caso de la figura 4.15.

Así pues, de nuevo el mejor caso se presenta en la figura 4.13, donde no existe oscilación de ningún tipo, se tiene otra vez un error máximo de 0.004 *rad.* aproximadamente. En el caso de que se estuviese dispuesto a lidiar con ligeras oscilaciones, sin embargo, la configuración más adecuada vendría dada por la 4.14, con ganancias PD de valores $K_p = 1500\boldsymbol{I}$ y $K_v = 150\boldsymbol{I}$.

En definitiva, con el control PD+ en simulación, se consigue un comportamiento casi idén-

tico al del control PD con prealimentación, por lo que las mismas conclusiones pueden ser sacadas en este caso en lo que a mejoría respecto al control por defecto se refiere, si bien es cierto que en una situación real, su desempeño debería diferenciarse más, como veremos en la sección 4.2.1.

4.1.1.3. Control Par-Calculado

En este apartado, se van a mostrar los resultados obtenidos para el controlador Par-Calculado, cuya ley de control se indica en la ecuación (3.14).

Como se indicó en su momento en el capítulo 3, el control Par-Calculado, a diferencia de los dos ya evaluados en este capítulo, no cuenta de forma específica con un término PD en su expresión, sino que éste viene multiplicado por la Matriz de Inercia $M(\boldsymbol{q})$, la cual depende de la posición actual del manipulador. En consecuencia, en este controlador las ganancias PD no son constantes al multiplicarse por dicho término, lo que provoca que en bucle cerrado este controlador no lineal pase a tener un comportamiento lineal, cosa que no sucede con el control PD+ ni con el control PD con prealimentación.

Como de costumbre en esta sección, se va a empezar la fase de experimentación con las ganancias más bajas, y de ahí se irán incrementando. Sin embargo, durante el proceso de pruebas se identificó que el rango de ganancias válidas para este controlador tenían valores más elevados que en el caso anterior. Esto, no obstante, es anecdótico, ya que los valores adoptados por las ganancias K_p y K_v dependen del controlador usado, del robot empleado, de la fidelidad del modelo dinámico o incluso de la trayectoria a seguir en algunos casos puntuales.

Sin más, en la figura 4.16 se muestran los resultados obtenidos con las ganancias más bajas.

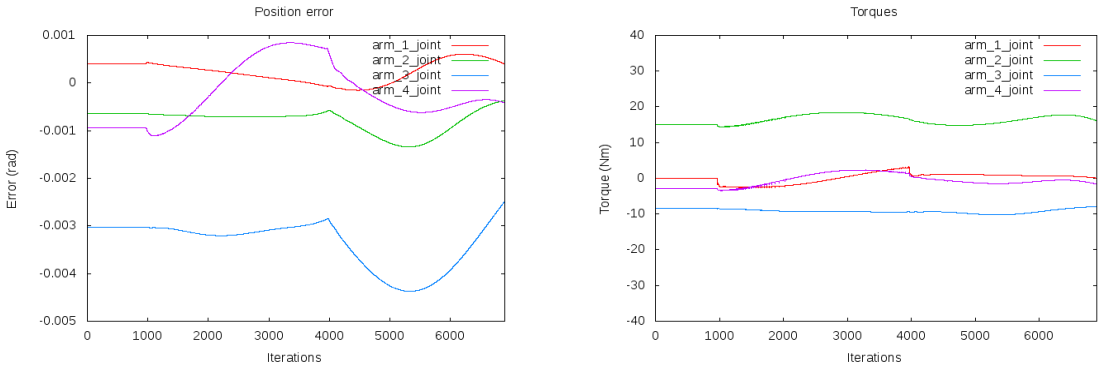


Figura 4.16: Control Par-Calculado. $K_p = 10000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

Como se puede observar de la figura 4.16, aun con los valores más bajos de ganancias que se van a tratar para este controlador, se consiguen mejores resultados que en los otros controladores presentados hasta el momento, sin rastro alguno de oscilación aparente, y con la única discrepancia de la articulación 3, de forma similar a como sucedía en los anteriores casos.

Así pues, antes de contrastar resultados, en las figuras 4.17 a 4.21 se van a mostrar los resultados del resto de pruebas, en las cuales se han seguido ajustando las ganancias PD del controlador con el fin de maximizar su eficacia.

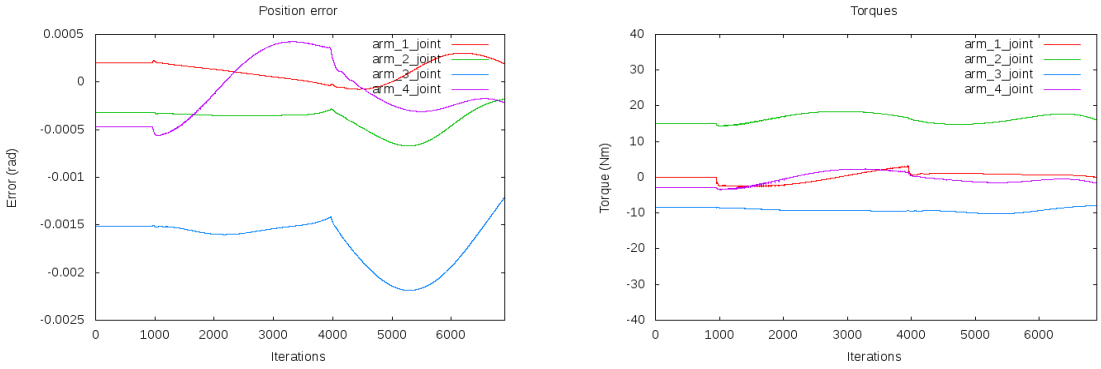


Figura 4.17: Control Par-Calculado. $K_p = 20000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

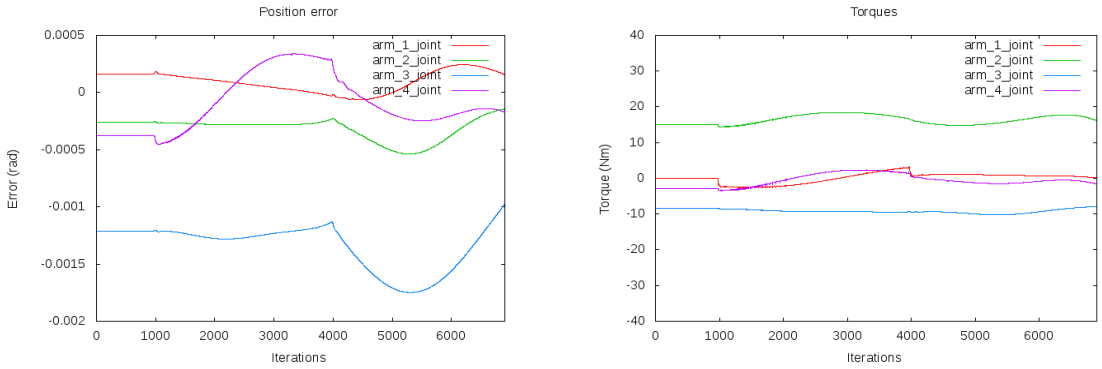


Figura 4.18: Control Par-Calculado. $K_p = 25000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

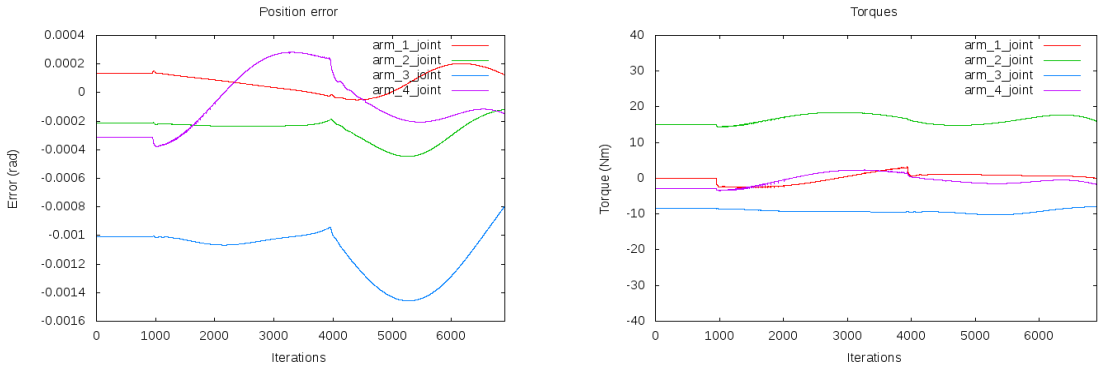


Figura 4.19: Control Par-Calculado. $K_p = 30000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

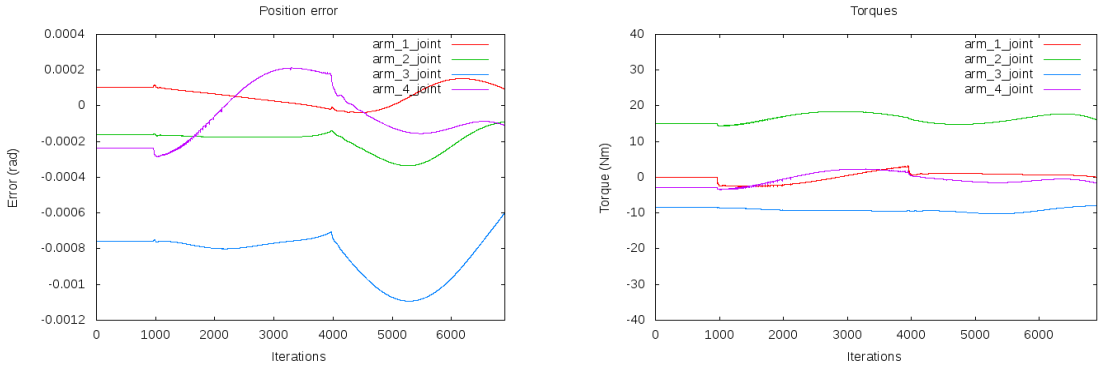


Figura 4.20: Control Par-Calculado. $K_p = 40000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

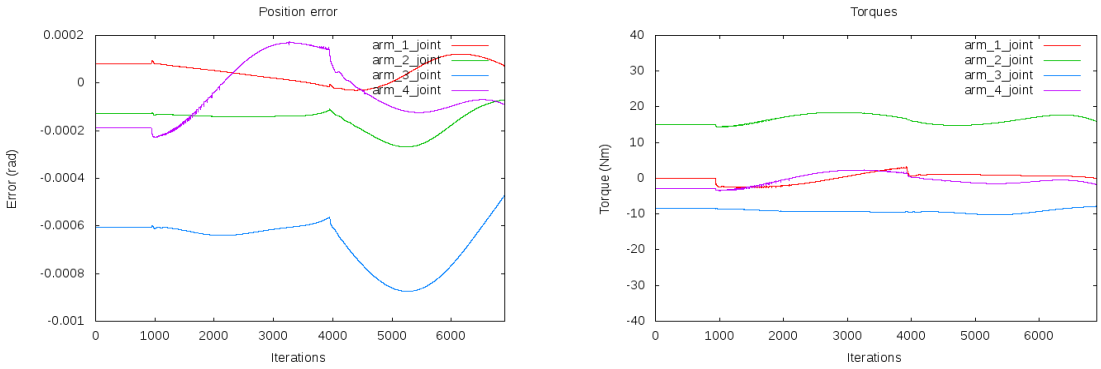


Figura 4.21: Control Par-Calculado. $K_p = 50000\boldsymbol{I}, K_v = 500\boldsymbol{I}$

Como se observa en las figuras anteriores, para este controlador se consiguen resultados mucho mejores, mejorando cada vez más conforme se van aumentando los valores de las ganancias PD. Esto se debe muy probablemente al hecho de que este controlador hace un uso mayor del modelo dinámico del robot, y dado que éste se encuentra en un entorno simulado, el modelo del robot describe exactamente su dinámica, teniendo una situación ideal que muy difícilmente se replicaría en la realidad.

Así pues, para este controlador el caso mejor corresponde con la última prueba, cuyos resultados se muestran en la figura 4.21. En este caso, el error mayor, correspondiente a la tercera articulación, no supera en valor absoluto los 0.001 *rad.*, mientras que lo más destacado son el resto de articulaciones, con errores oscilando entre $\pm 0.0002 rad$. Sin embargo, como se ha comentado, estos resultados son en cierta forma engañosos, puesto que son consecuencia de hacer uso de un modelo dinámico perfecto del robot, cosa bastante inverosímil en la práctica.

Por otro lado, resulta curioso como en este caso no se llega a un comportamiento oscilatorio en ningún momento, lo cual viene dado por dos factores: El propio controlador y su estructura, unido a la retención de la ganancia K_d en el mismo valor. Se tomó esta decisión dado que es conocido que esta ganancia no debe tomar valores demasiado altos ya que puede llegar a inestabilizar el sistema. Por esto, como ya de por sí los valores de las ganancias debían ser elevados, se mantuvo fijo este valor para no llegar a dicha inestabilidad. En la figura 4.22, se puede ver como aumentando en demasía la ganancia K_v, se introduce comportamiento

inestable pese a reducir consecuentemente la ganancia K_p.

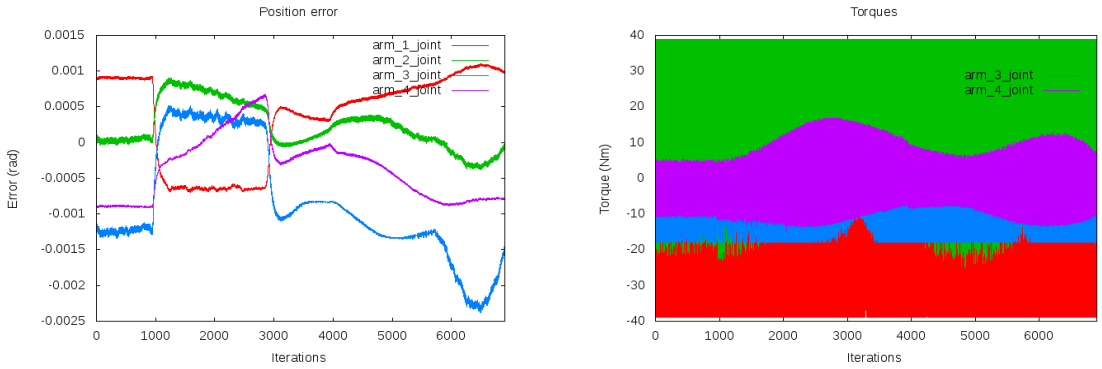


Figura 4.22: Control Par-Calculado. $K_p = 30000\boldsymbol{I}, K_v = 2000\boldsymbol{I}$

En definitiva, con el control Par-Calculado en simulación, se consigue el mejor comportamiento de entre los controladores articulares estudiados en este proyecto. Sin embargo, tanto el hecho de que se esté partiendo de un modelo dinámico perfecto, así como el alto valor de las ganancias, hacen presuponer que en el robot real este controlador no funcionará tan correctamente. Aun así, se han conseguido buenos resultados, que en el caso de disponer de un modelo dinámico lo más fiel posible no deberían distar en demasía de la realidad.

4.1.2. Control dinámico cartesiano

Una vez expuesta la experimentación llevada a cabo con los controladores articulares implementados para este proyecto, a continuación se van a exponer los resultados obtenidos para los controladores cartesianos, tanto el basado en Par-Calculado como el controlador óptimo.

Así pues, para estos controladores se va a seguir el mismo proceso que en el caso de los articulares, mostrando el error en posición y los pares articulares ejercidos en cada una de las pruebas llevadas a cabo para cada controlador. Sin embargo, al tratarse de controladores cartesianos, el error en posición vendrá dado en coordenadas cartesianas XYZ, en lugar de en radianes como se venía haciendo hasta el momento.

En lo que respecta a la trayectoria a seguir, será la misma que en el caso de los articulares, pero obteniendo en cada iteración del bucle de control las coordenadas cartesianas deseadas para el extremo del brazo aplicando la cinemática direca a la consigna en coordenadas articulares que nos proporciona el planificador, tal y como se adelantó en la sección 3.3.2. No obstante, este hecho no afecta en nada al controlador, que sigue recibiendo igualmente referencias cartesianas, independientemente de como sean generadas.

4.1.2.1. Control cartesiano basado en Par-Calculado

Tal y como se estudió en el capítulo 3, este controlador no es más que una extensión del control Par-Calculado, añadiendo la teoría de la cinemática diferencial que relaciona magnitudes articulares y cartesianas a través de la matriz Jacobiana. Así pues, en este apartado se van a exponer los resultados obtenidos usando la ley de control expresada en la ecuación (3.23).

Del mismo modo que con los controladores articulares, se mostrarán los resultados obtenidos de la experimentación de menor a mayor valor de las ganancias PD, teniendo en cuenta que para los controladores cartesianos las ganancias afectan a las diferentes coordenadas cartesianas $(x, y, z, \alpha, \beta, \gamma)$, y no directamente a la propia articulación, como en los controladores anteriores. Además, como este controlador deriva del control Par-Calculado, de igual modo las ganancias empleadas serán considerables, aunque no tanto como en el caso del controlador articular.

Así pues, en la figura 4.23 se muestran los resultados obtenidos con la primera prueba realizado.

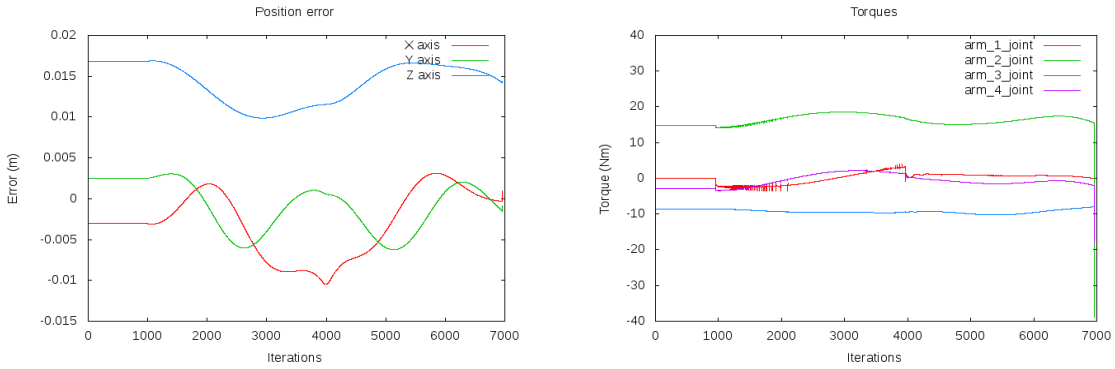


Figura 4.23: Control cartesiano basado en Par-Calculado. $K_p = 1000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

Como se puede observar en la figura 4.23, el resultado obtenido es bastante positivo aun en la primera prueba, puesto que el error máximo corresponde al eje Z con un valor de tan solo 1.5 cm, mientras que en los ejes XY apenas se roza el centímetro de error. Por su parte, los pares articulares de nuevo son moderados, sin acercarse demasiado a sus valores máximos.

Antes de seguir con la valoración de los resultados, se van a mostrar el resto de experimentos llevados a cabo con este controlador, correspondientes a las figuras 4.24 a 4.28 con el fin de comprobar hasta qué punto se puede maximizar la eficiencia de éste y reducir aun más el error cartesiano.

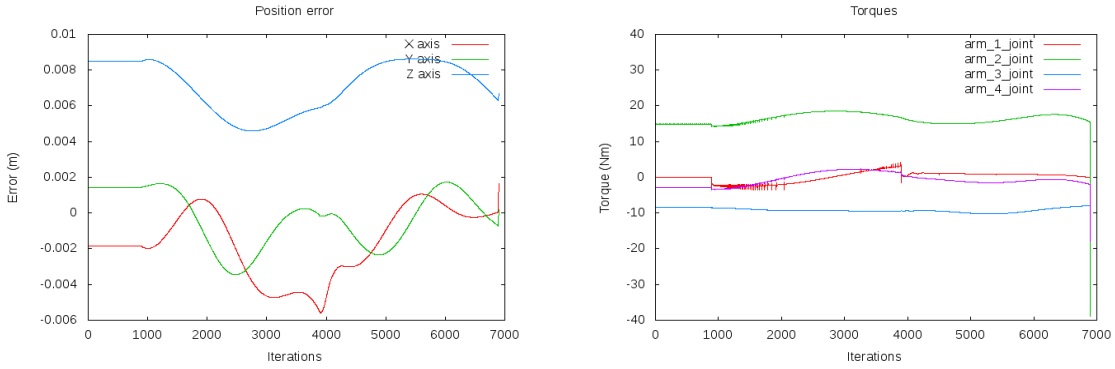


Figura 4.24: Control cartesiano basado en Par-Calculado. $K_p = 2000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

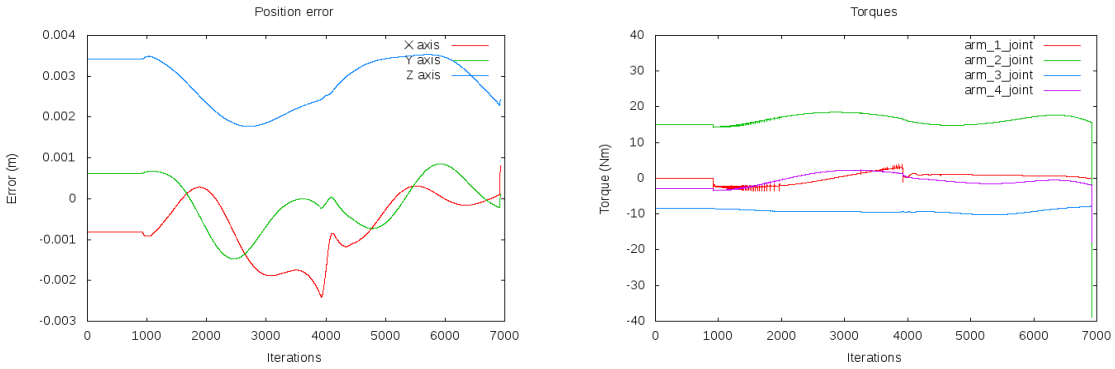


Figura 4.25: Control cartesiano basado en Par-Calculado. $K_p = 5000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

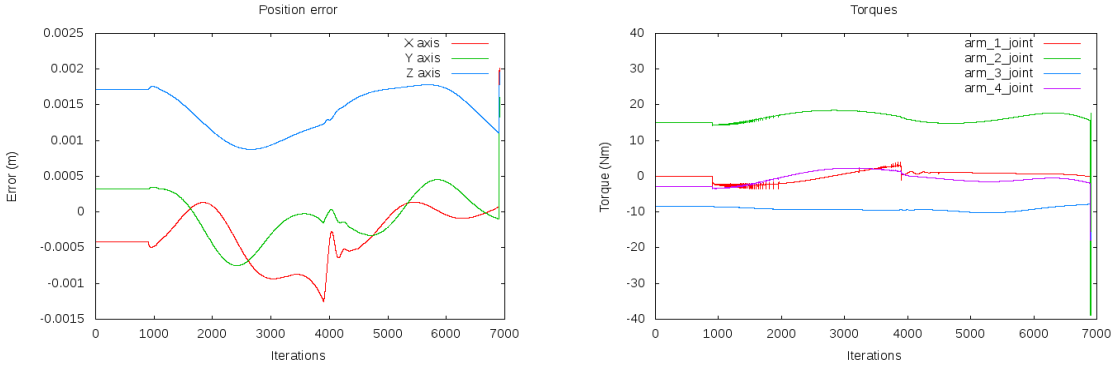


Figura 4.26: Control cartesiano basado en Par-Calculado. $K_p = 10000\boldsymbol{I}, K_v = 200\boldsymbol{I}$

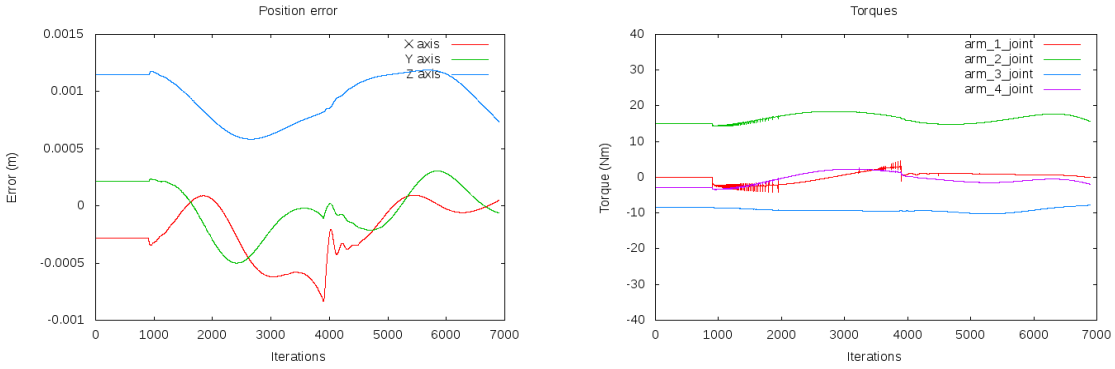


Figura 4.27: Control cartesiano basado en Par-Calculado. $K_p = 15000\boldsymbol{I}, K_v = 300\boldsymbol{I}$

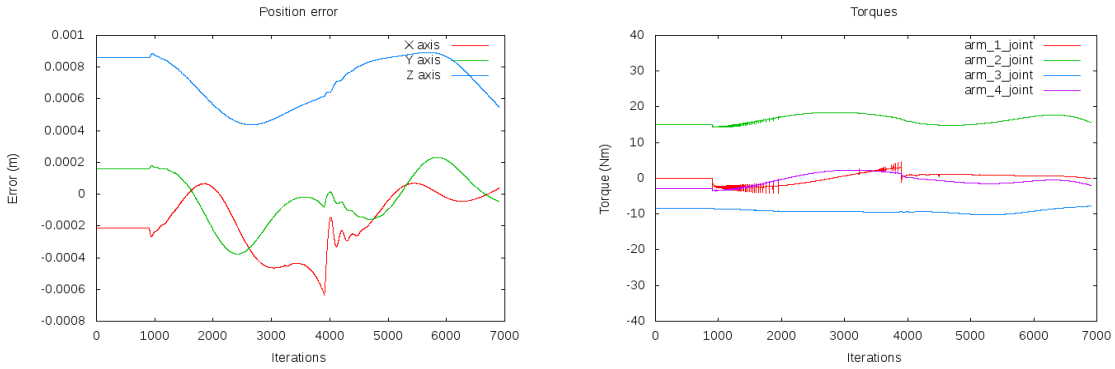


Figura 4.28: Control cartesiano basado en Par-Calculado. $K_p = 20000\boldsymbol{I}, K_v = 300\boldsymbol{I}$

Observando las figuras anteriores, se comprueba como se obtienen buenos resultados en general, mejorando a la vez que se incrementan los valores de las ganancias PD. De nuevo, al igual que sucedía con el controlador Par-Calculado articular, estos resultados probablemente se deban al hecho de estar en un entorno simulado, sin perturbaciones y con un modelo dinámico idéntico al manipulador simulado, por lo que se deben tomar con cautela estos resultados.

Así pues, se puede apreciar como la configuración del controlador propuesta en la figura 4.28 ofrece los mejores resultados, con un error máximo en el eje Z cercano al milímetro, mientras que en los ejes XY apenas se alcanzan valores de 0.2 milímetros, obteniendo así una alta precisión. Además, de nuevo conteniendo los incrementos de la ganancia K_v conseguimos evitar que se alcance un comportamiento inestable del sistema, ya que si aumentásemos K_v en proporción a K_p, tendríamos inestabilidades como la expuesta en la figura 4.29.

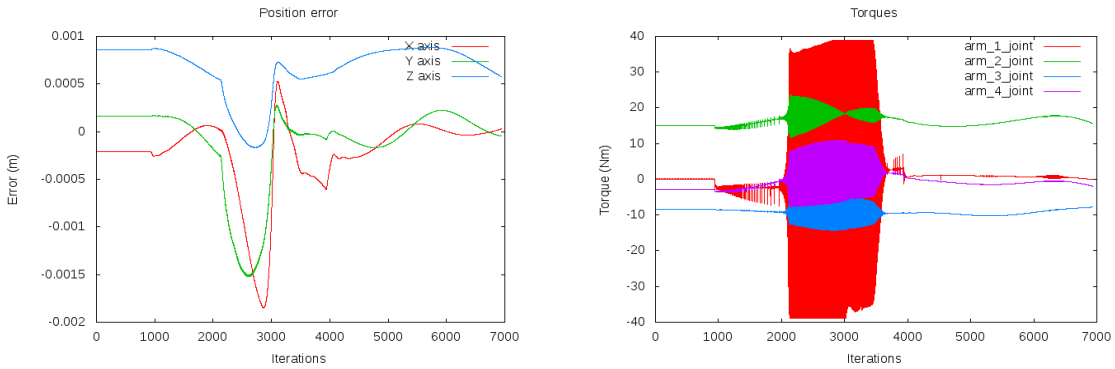


Figura 4.29: Control cartesiano basado en Par-Calculado. $K_p = 20000\boldsymbol{I}, K_v = 800\boldsymbol{I}$

En definitiva, con el control cartesiano basado en Par-Calculado en simulación, se consigue un seguimiento muy preciso de la trayectoria cartesiana deseada, con errores inferiores al milímetro en el mejor de los casos. Sin embargo, al igual que en el caso articular, el hecho de que se esté partiendo de un modelo dinámico perfecto, así como el elevado valor de las ganancias, llevan a pensar que en la práctica no se sostendrá este nivel de precisión. Aun así, los resultados conseguidos son buenos, y en un robot real cuanto más fiel sea el modelo del robot, más posible será mantener un nivel de precisión similar.

4.1.2.2. Control óptimo

Por último dentro de la sección de este capítulo dedicada a la experimentación en simulación de los controladores dinámicos multiarticulares implementados, en este apartado se van a exponer los resultados obtenidos para el controlador óptimo cartesiano, cuya ley de control viene dada por la ecuación (3.32).

Tal y como se explicó en el capítulo 3, sección 3.2, el control óptimo aplicado a robots manipuladores tiene como objetivo aprovechar la redundancia del robot con el fin de llevar a cabo el seguimiento de una trayectoria a la vez que se trata de optimizar una determinada función de coste. Esta optimización viene dada por el valor de la matriz $\boldsymbol{W}$ en la ecuación (3.32), ya que según el valor que tome, se producirá un efecto u otro sobre el robot.

Así pues, tras varias pruebas iniciales, se decidió emplear para este proyecto el valor $\boldsymbol{W} = M^{-1}$. Esta decisión se tomó en parte basándose en los resultados obtenidos en [32] y [33], ya que en ambos casos los mejores resultados se obtuvieron con este valor de $\boldsymbol{W}$. Además, en el caso de $\boldsymbol{W} = M^{-2}$, el controlador obtenido es idéntico al control cartesiano basado en Par-Calculado, por lo que repetiríamos los resultados anteriores. Igualmente, la elección del valor $\boldsymbol{W} = M^{-1}$ no es en absoluto arbitraria, puesto que es consistente con el principio de d'Alembert [64] sobre el trabajo virtual.

Con esto, y al igual que en el resto de controladores, se van a exponer los resultados obtenidos con este controlador empezando por el caso de ganancias menores, como se muestra

en la figura 4.30.

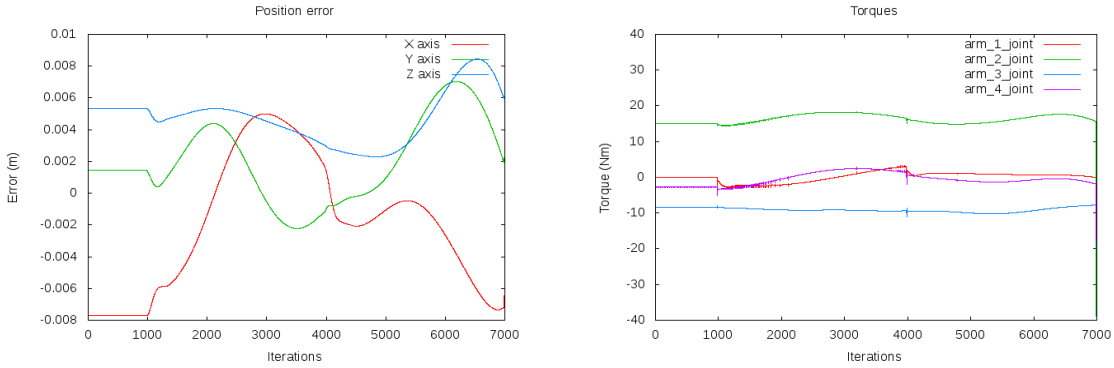


Figura 4.30: Control óptimo cartesiano. $K_p = 2000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

Tal y como se muestra en la figura 4.30, para el caso de ganancias más bajas se tiene un error máximo similar al caso del control cartesiano basado en Par-Calculado, ligeramente inferior puesto que no se llega al centímetro de error. Por otro lado, en lo que a los pares articulares respecta, en este caso no se aprecia la optimización en éstos, probablemente al tratarse de un entorno simulado.

No obstante, se van a mostrar el resto de resultados obtenidos, correspondientes a las figuras 4.31 a 4.35 antes de llevar a cabo cualquier análisis, con el fin de comprobar hasta qué punto se puede reducir el error cartesiano en el seguimiento de la trayectoria, y si se produce una reducción más visible de los pares articulares ejercidos en el robot.

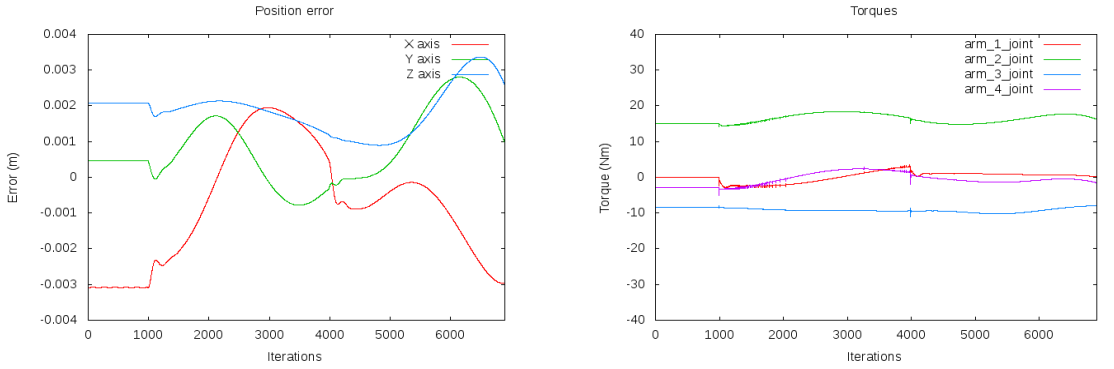


Figura 4.31: Control óptimo cartesiano. $K_p = 5000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

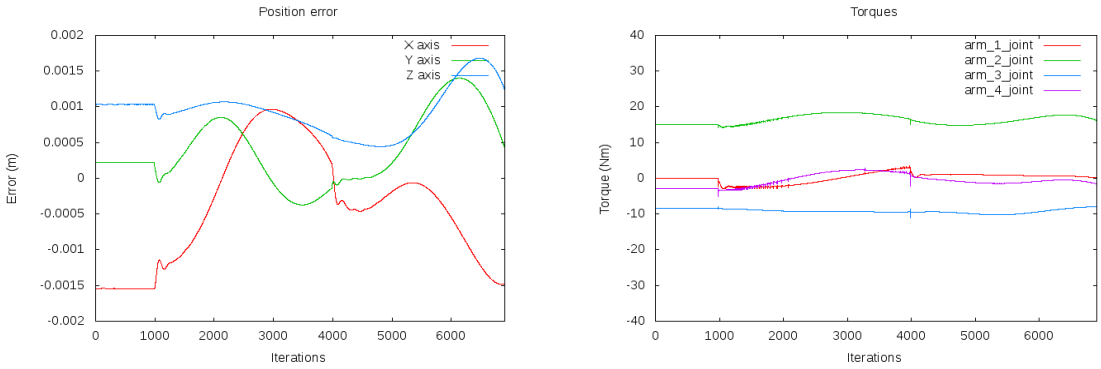


Figura 4.32: Control óptimo cartesiano. $K_p = 10000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

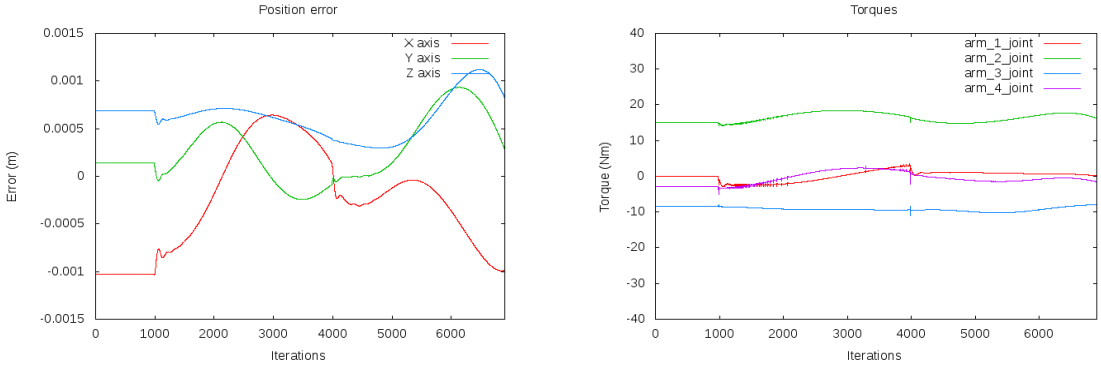


Figura 4.33: Control óptimo cartesiano. $K_p = 15000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

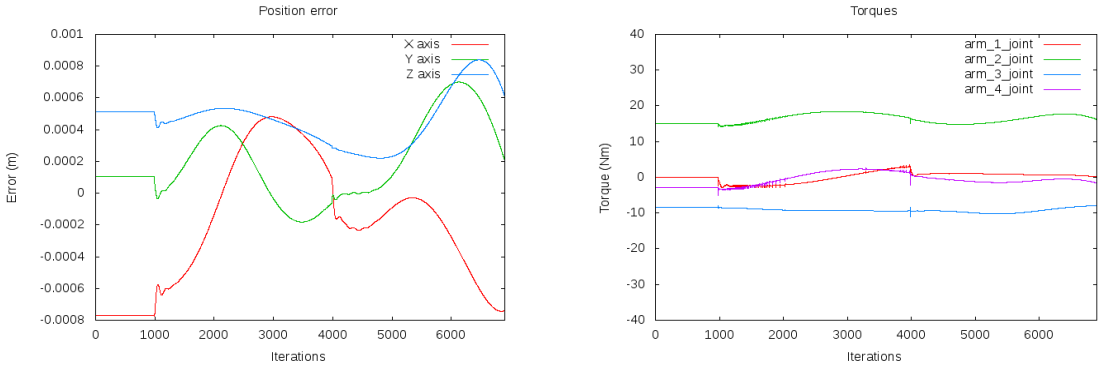


Figura 4.34: Control óptimo cartesiano. $K_p = 20000\boldsymbol{I}, K_v = 100\boldsymbol{I}$

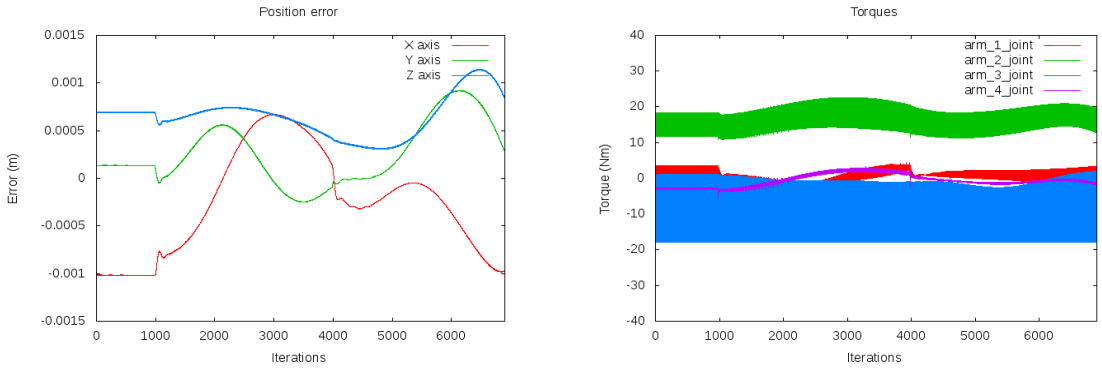


Figura 4.35: Control óptimo cartesiano. $K_p = 15000\boldsymbol{I}, K_v = 150\boldsymbol{I}$

Tal y como se desprende de las figuras anteriores, se aprecia como los resultados de nuevo vuelven a ser correctos, similares al caso anterior pero ligeramente inferiores, reduciéndose el error conforme se incrementan los valores de las ganancias PD.

Así pues, la configuración del controlador propuesta en la figura 4.34 arroja los mejores resultados, con un error máximo en el eje Z inferior al milímetro, mientras que en los ejes XY se oscila entre valores cercanos a los 0.5 milímetros, lo cual indica una precisión bastante elevada. Además, en este caso en una de las pruebas se optó por aumentar ligeramente el valor de la ganancia K_v, aun reduciendo el valor de K_p para intentar evitar un comportamiento inestable. Este hecho se ilustra en la figura 4.35, donde se puede observar que este cambio ha introducido un ligero comportamiento oscilatorio en los pares articulares, además de provocar saturación de par en la articulación 3, lo cual, a pesar de no llevar a inestabilidad, ha causado un aumento del error en posición.

En conclusión, se ha comprobado como con el controlador óptimo en simulación se mejora, si bien tan solo ligeramente, el seguimiento de la trayectoria por parte del robot, consiguiendo reducir tanto el error en posición medio como el par articular ejercido por los actuadores. Este hecho debiera verse más enfatizado en el caso del robot real al no contar con un modelo dinámico tan fiel, lo que reduciría el desempeño del control cartesiano basado en Par-Calculado.

Así pues, se va a proceder en la siguiente sección a analizar los resultados obtenidos en las

pruebas llevadas a cabo con el robot TIAGo en la sede de PAL Robotics.

4.2. Robot TIAGo

En esta sección, una vez se han presentado los resultados obtenidos en entorno simulado con los controladores dinámicos multiarticulares desarrollados en el marco de este proyecto, se va a mostrar el comportamiento de dichos controladores aplicados al robot TIAGo real.

Con este fin, se realizó una estancia de tres días en las instalaciones de la compañía PAL Robotics, de la mano del co-tutor de este proyecto, Jordi Pagès. En este tiempo, se consiguió adaptar el software que implementa los controladores para hacerlos funcionar en la plataforma real, y se realizaron una gran cantidad de pruebas con distintas configuraciones de ganancias PD, de entre las cuales se han extraído las más representativas para ser incluidas en esta memoria.

Así pues, en esta sección se seguirá una estructura similar a la sección 4.1. En primer lugar, se mostrarán los resultados obtenidos para los controladores dinámicos articulares, analizando y comparando brevemente su desempeño, cosa que se hará más en detalle en la próxima sección 4.3.

Tras esto, se expondrá el comportamiento del robot haciendo uso de los controladores dinámicos cartesianos implementados, haciendo especial hincapié en la comparativa entre éstos, con el fin de comprobar si efectivamente el controlador óptimo mejora al control basado en Par-Calculado.

Por último, durante esta estancia se implementó también un controlador visual óptimo basado en posición, puesto que como se ha indicado en el capítulo 3, sección 3.2.2, al no contar el robot con una cámara en su manipulador, no se pudo aplicar directamente el controlador obtenido en dicha sección, cuya ley de control viene dada en (3.44). Este controlador será expuesto en la sección 4.2.3.

Así pues, antes de mostrar los resultados obtenidos, cabe mencionar el proceso de adaptación del software en el que se implementaron los controladores, así como los cambios del robot real respecto al robot simulado.

Uno de los cambios más destacados es el hecho de que en la simulación efectivamente se realiza un control directo, siendo la acción de control enviada al robot los pares articulares de las articulaciones controladas en cada instante de la trayectoria. Por contra, en el caso del robot real el control se lleva a cabo por corriente, siendo necesario por tanto realizar la conversión de par articular a corriente antes de enviar la acción de control. Dicha conversión viene dada por la ecuación (4.1).

$$I = \frac{\tau}{(k_m R)}, \tag{4.1}$$

donde I es la intensidad de corriente enviada finalmente al motor, τ es la acción de control obtenida por el controlador, R es el ratio de reducción del motor de la articulación y k_m una constante proporcionada por el fabricante en las especificaciones del actuador.

El otro problema principal es el hecho de que dejamos de tener un modelo dinámico fiel a la realidad. De hecho, en el caso del robot real el modelo dinámico obtenido con KDL del URDF no tiene en consideración componentes de fricción, inercias de los motores, o parámetros de elasticidad propios de las reductoras *Harmonic Drive* empleadas en el TIAGo.

Por último, cabe destacar que el bucle de control en el robot real se ejecuta con una frecuencia de 100 Hz, a diferencia de los 1000 Hz con los que funciona en simulación. Esta frecuencia es bastante más baja de lo recomendado para realizar un control directo de las articulaciones.

Así pues, con estas consideraciones se puede intuir que los resultados en el robot real distarán de los obtenidos en simulación, aunque esto es un hecho que se analizará y justificará en la sección 4.3.

Sin más, a continuación se muestran los resultados obtenidos en la experimentación reali-

zada con el robot real.

4.2.1. Control dinámico articular

En este apartado, se van a presentar los resultados más representativos de entre los obtenidos en las pruebas realizadas con el robot real para los controladores articulares implementados.

Para esto, se hizo uso de un robot TIAGo *Steel*, con una pinza de dos dedos como efector final, aunque esta parte no era controlada. En cuanto al entorno donde se llevo la experimentación, este se puede apreciar en la figura 4.36.

Figura 4.36: Entorno para pruebas en PAL Robotics.

En la figura 4.36 se ve como el robot se encuentra ya en la posición inicial de la trayectoria descrita en la sección 4.1, aunque en este caso se extendió hasta 14 segundos su duración, y

cómo dispone de amplio espacio a su alrededor para realizarla sin problemas. Cabe destacar que en dicha imagen el robot ya tenía en funcionamiento uno de los controladores implementados, por lo que se puede observar como se lleva a cabo perfectamente la compensación de gravedad, así como la estabilidad para mantener la posición actual.

Con esto, a continuación se van a mostrar los resultados de los controladores implementados y adaptados al robot real. Dado que en los apartados correspondientes a cada controlador pertenecientes a la sección 4.1 ya se recordaron sus características principales, en este caso se obviarán estas partes y se mostrarán directamente los resultados obtenidos.

4.2.1.1. Control PD con precompensación

En este punto, se van a presentar los resultados obtenidos en el robot real para el control PD con prealimentación (3.3), los cuales se muestran en las figuras 4.37 a 4.39.

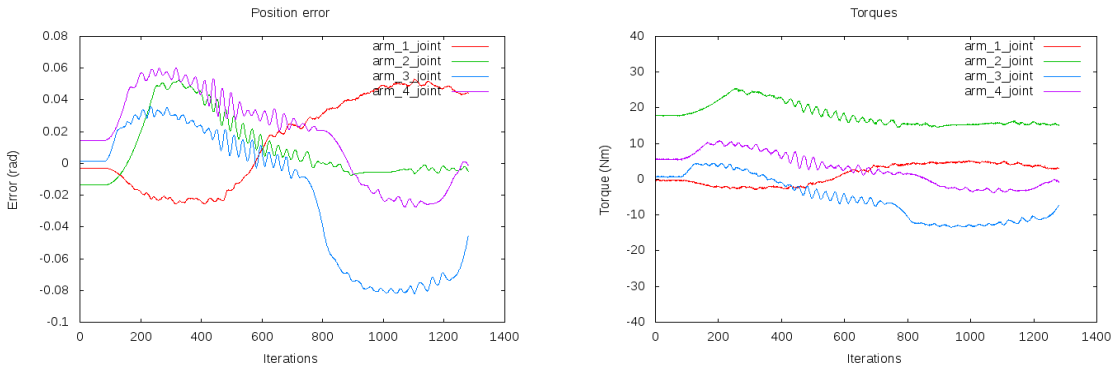


Figura 4.37: Control PD con prealimentación en robot real. $K_p = 100\boldsymbol{I}, K_v = 20\boldsymbol{I}$

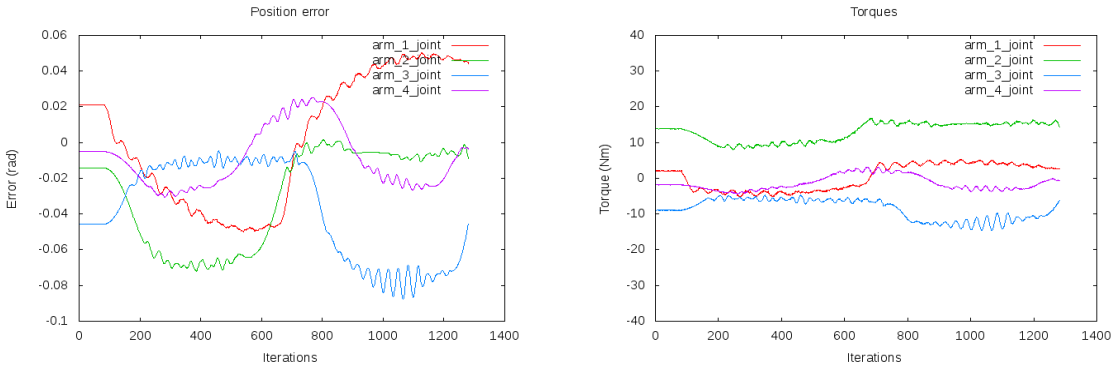


Figura 4.38: Control PD con prealimentación en robot real. $K_p = 100\boldsymbol{I}, K_v = 30\boldsymbol{I}$

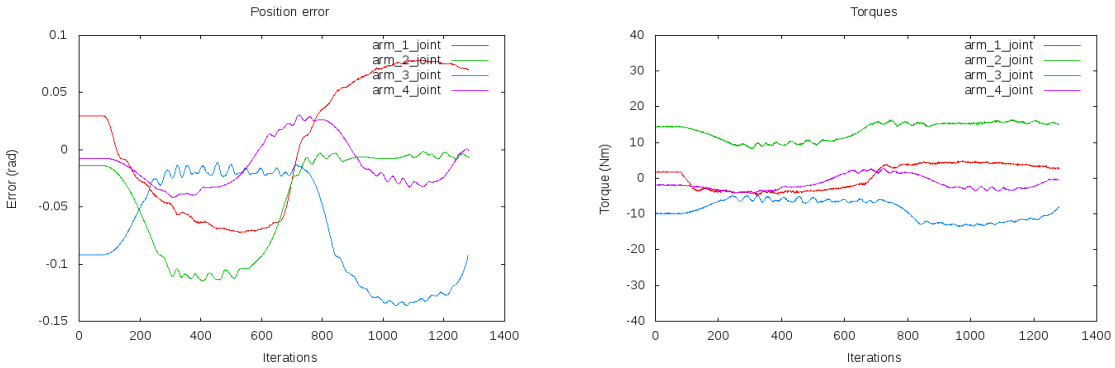


Figura 4.39: Control PD con prealimentación en robot real. $K_p = diag(60, 60, 60, 80), K_v = 20\boldsymbol{I}$

Como se puede observar en las figuras anteriores, el comportamiento del controlador es, evidentemente, peor que en entorno simulado, puesto que en el robot real contamos con un modelo dinámico imperfecto, así como con los fallos propios de hacer uso de componentes físicos: Perturbaciones, imperfecciones, rozamientos, etc.

Otro hecho destacable es la presencia de pequeñas oscilaciones en los resultados obtenidos. Este hecho viene motivado principalmente por la escasa frecuencia a la que se ejecuta el bucle de control, lo que implica que se actualice la acción de control cada 10 ms, lo cual es demasiado tiempo en control directo, y tiene como consecuencia la aparición de estas oscilaciones.

Así pues, se puede observar que el mejor comportamiento se obtiene con la sintonización de ganancias de la figura 4.38, donde se tiene un pico de error máximo de cerca de 0.08 *rad.*,

lo que equivale a 4.5 grados en el sistema sexagesimal. Por otro lado, los pares articulares se mantienen casi sin oscilaciones, además de estar bastante lejos de sus valores límite.

En definitiva, el comportamiento obtenido con este controlador implantado en el robot real arroja resultados satisfactorios. Obviamente, estos no llegan al nivel de los obtenidos en simulación, sin embargo este hecho era algo anticipado, puesto que siempre que se da el salto a trabajar con componentes físicos surgen discrepancias y problemas que no aparecen en la situación ideal de la simulación.

4.2.1.2. Control PD+

En este apartado, se va a proceder a la presentación de los resultados obtenidos en el robot real para el control PD+ (3.8). Dichos resultados están representados en las figuras 4.40 a 4.42.

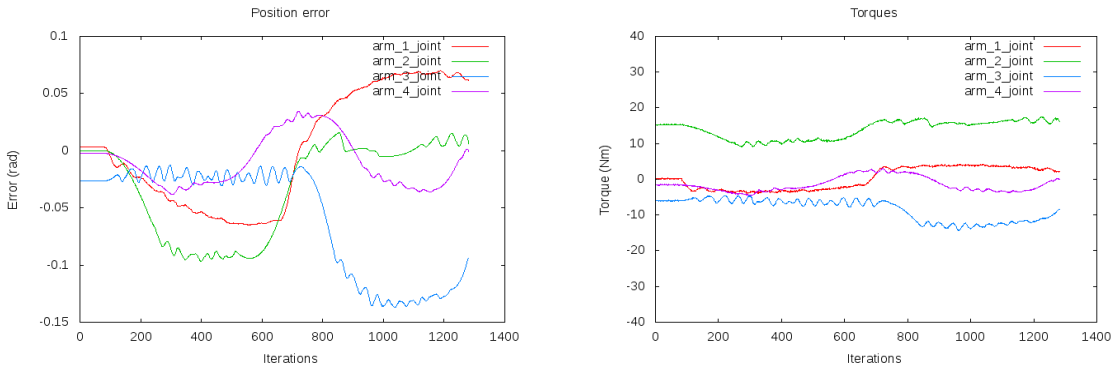


Figura 4.40: Control PD+ en robot real. $K_p = diag(60, 60, 60, 80)$, $K_v = 20\boldsymbol{I}$

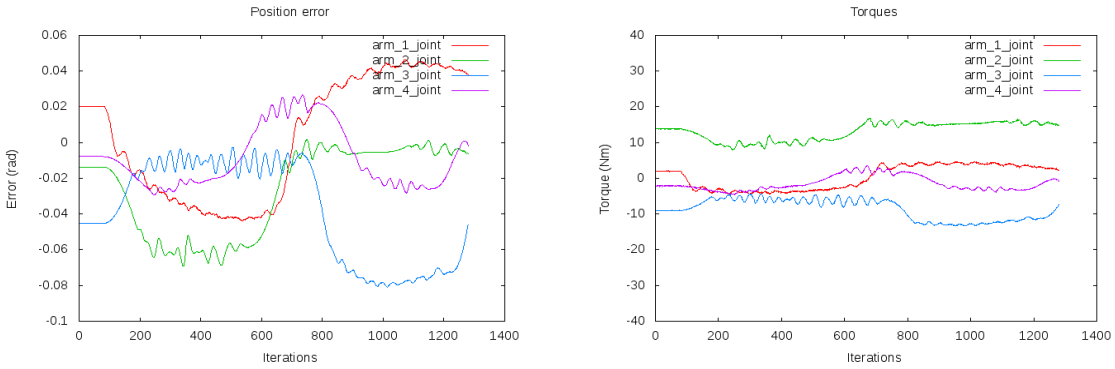


Figura 4.41: Control PD+ en robot real. $K_p = 100\boldsymbol{I}, K_v = 30\boldsymbol{I}$

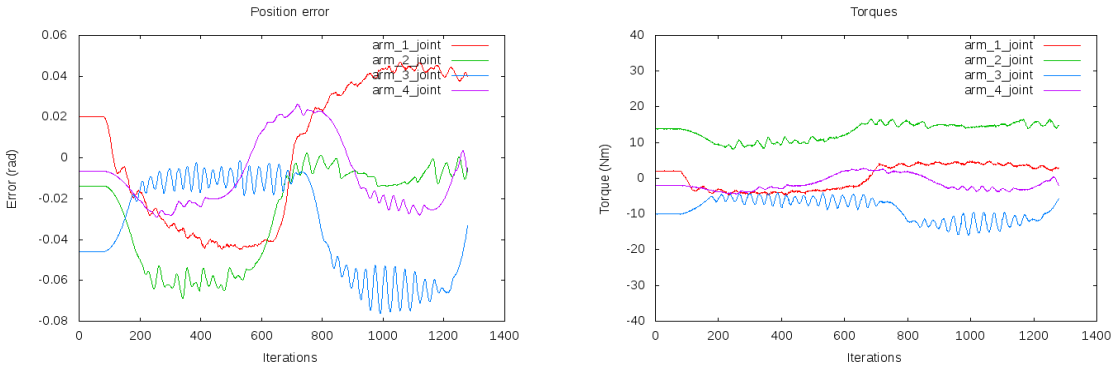


Figura 4.42: Control PD+ en robot real. $K_p = diag(100, 100, 120, 100), K_v = diag(20, 20, 30, 20)$

Tal y como se muestra en las figuras anteriores, el comportamiento del controlador al igual que en el caso del control PD con prealimentación es bastante correcto, teniendo en cuenta las limitaciones de hardware que se tienen, no habiendo oscilaciones excesivas en ningún caso, y con un error en posición aceptable.

Al igual que en la sección 4.1, los resultados obtenidos con el control PD+ son similares al control PD con prealimentación debido a la gran similitud entre ambos. Sin embargo, en este caso sí que se aprecia una ligera diferencia a favor del control PD+, tanto por tener un poco menos de error como por ser menos oscilante en el caso mejor.

Así pues, se puede apreciar que el mejor comportamiento obtenido en lo que a error en posición máximo se refiere aparece con configuración de la prueba ilustrada en la figura 4.42,

donde se tiene un pico de error máximo inferior a los 0.08 *rad*.. Sin embargo, en este caso aparecen ligeras oscilaciones en la articulación 3, por lo que en caso de querer evitarlas, la opción más adecuada sería la de la figura 4.41, ya que apenas aumenta el error en posición, pero sí que reduce mucho la oscilación en la articulación mencionada.

En conclusión, del comportamiento obtenido con el control PD+ aplicado al robot real se obtienen resultados correctos, mejorando ligeramente los obtenidos con el control PD con prealimentación.

4.2.1.3. Control Par-Calculado

En este punto, se va a llevar a cabo la exposición de la experimentación realizada en el robot real para el control Par-Calculado (3.14), cuyos resultados de error en posición y par articular se ilustran en las figuras 4.43 a 4.45.

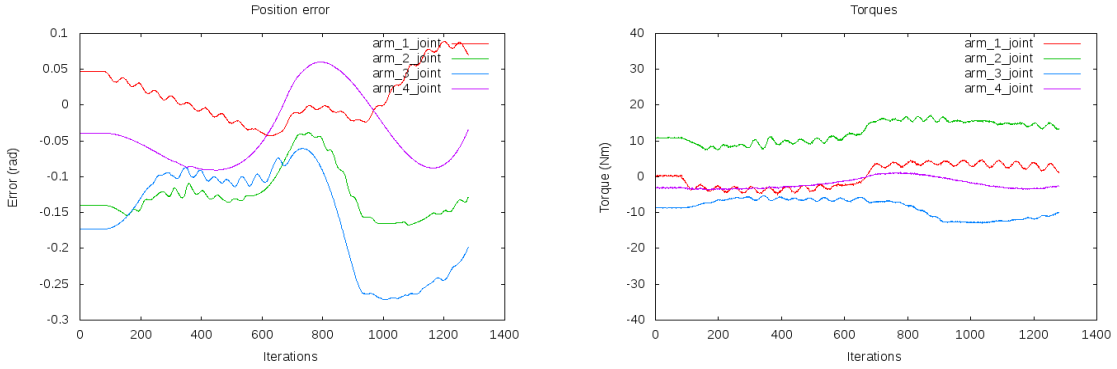


Figura 4.43: Control Par-Calculado en robot real. $K_p = diag(100, 200, 400, 200), K_v = diag(20, 28, 35, 20)$

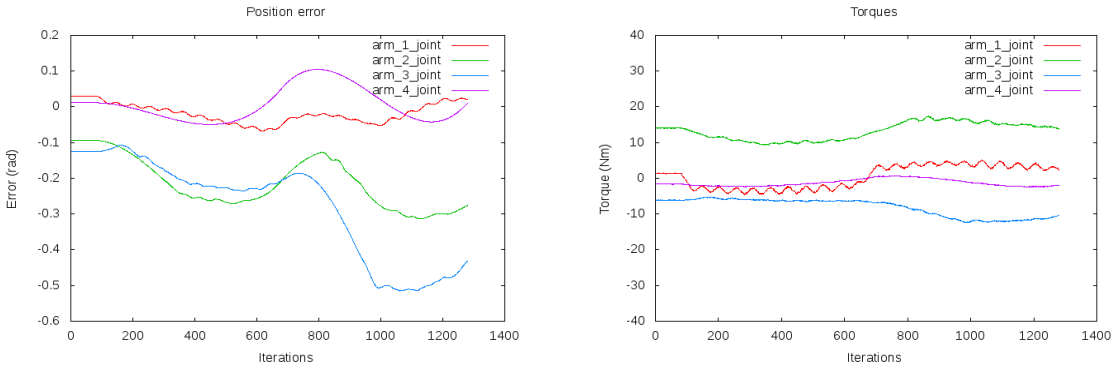


Figura 4.44: Control Par-Calculado en robot real. $K_p = diag(100, 100, 200, 100), K_v = diag(20, 20, 30, 20)$

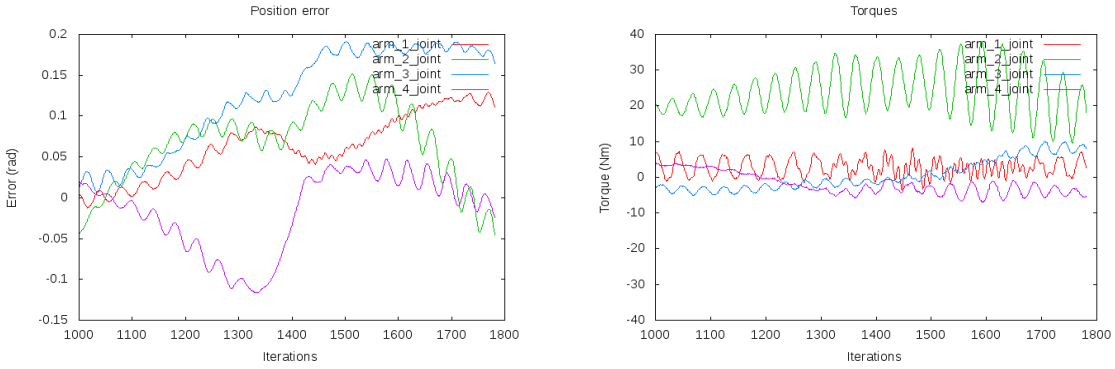


Figura 4.45: Control Par-Calculado en robot real. $K_p = diag(200, 150, 600, 600), K_v = diag(80, 120, 85, 85)$

En este caso, como se extrae de las figuras anteriores, el comportamiento del controlador es bastante peor que en simulación, con resultados que distan considerablemente de los obtenidos en el entorno simulado.

Sin embargo, esto es algo que ya se anticipó cuando se mostraron los resultados de este mismo controlador en entorno simulado. El control Par-Calculado depende en gran medida de la fidelidad del modelo dinámico del robot, así como de una frecuencia de actualización de la acción de control más elevada. Estos dos factores son precisamente los que más fallan en el robot real, pensado para un control en posición, en el cual dichos factores no son tan relevantes.

Así pues, en este caso observamos que el mejor comportamiento se tiene en la figura 4.43, ya que apenas se tienen oscilaciones, y aunque el error máximo es cercano a los 0.3 *rad.*, esto sucede en un momento puntual, ya que el error medio se encuentra en aproximadamente 0.15 *rad.*. Por otro lado, en la figura 4.45 se tiene tanto un error máximo como un error medio inferiores a la figura 4.43, sin embargo se observa en los pares articulares un comportamiento oscilante considerable, por lo que probablemente en otras trayectorias se llegaría a un comportamiento inestable del robot.

En definitiva, este controlador ofrece los peores resultados dentro de los controladores articulares aplicados al robot real, lo cual viene dado por el uso de un modelo dinámico poco detallado, así como una frecuencia en el ciclo de control demasiado baja para un controlador de estas características.

4.2.2. Control dinámico cartesiano

Una vez mostrados los resultados de la experimentación llevada a cabo con los controladores articulares implementados, a continuación se procederá a exponer los resultados obtenidos con los controladores dinámicos cartesianos desarrollados en este proyecto tras las pruebas realizadas en el robot real, con el fin de comprobar su desempeño.

El proceso a seguir será idéntico al realizado en la sección 4.1, mostrando las gráficas de error en posición cartesiana y pares articulares ejercidos para cada una de las pruebas realizadas. Además, mantendremos la misma trayectoria para así poder comparar eficazmente los resultados obtenidos en el robot real con los analizados para el caso del entorno simulado.

4.2.2.1. Control cartesiano basado en Par-Calculado

En este apartado, se mostrarán los resultados obtenidos al implantar en el robot TIAGo el control cartesiano basado en Par-Calculado (3.23) implementando en este proyecto, pudiéndose observar los valores de error en posición y par articular de las pruebas más características representados en las figuras 4.46 a 4.48.

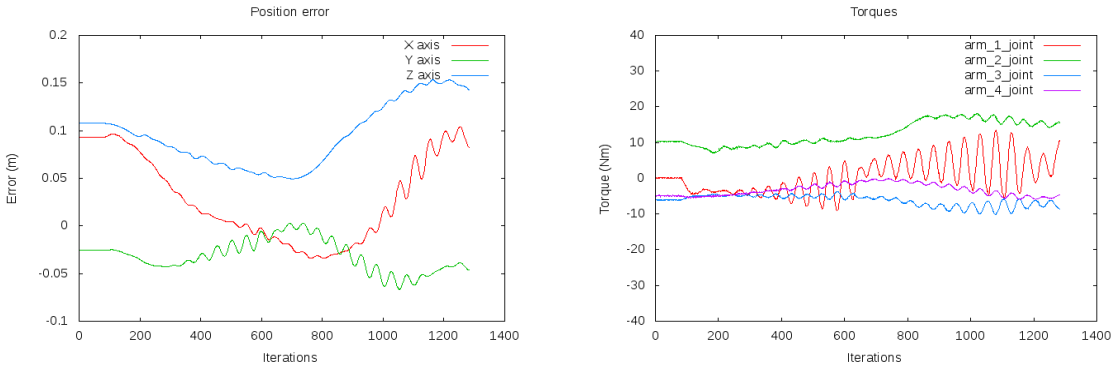


Figura 4.46: Control cartesiano basado en Par-Calculado en robot real. $K_p = diag(400, 400, 400, 150, 150, 150), K_v = 20\boldsymbol{I}$

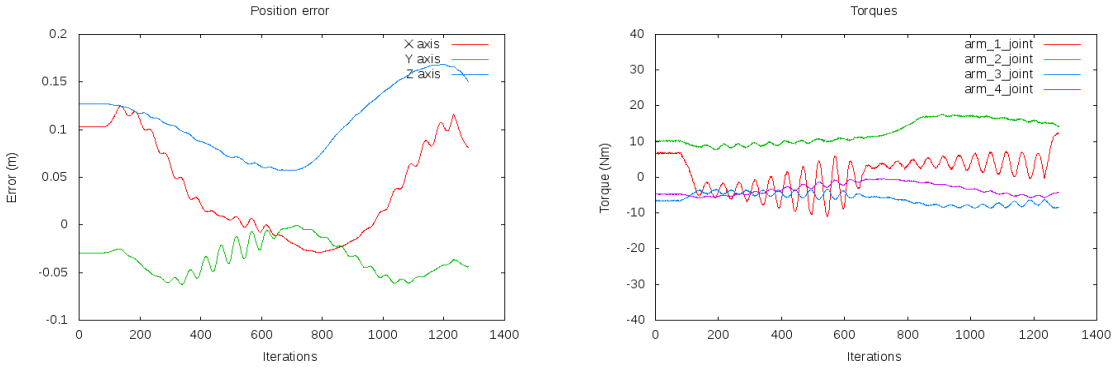


Figura 4.47: Control cartesiano basado en Par-Calculado en robot real. $K_p = diag(300, 300, 300, 150, 150, 150), K_v = 20\boldsymbol{I}$

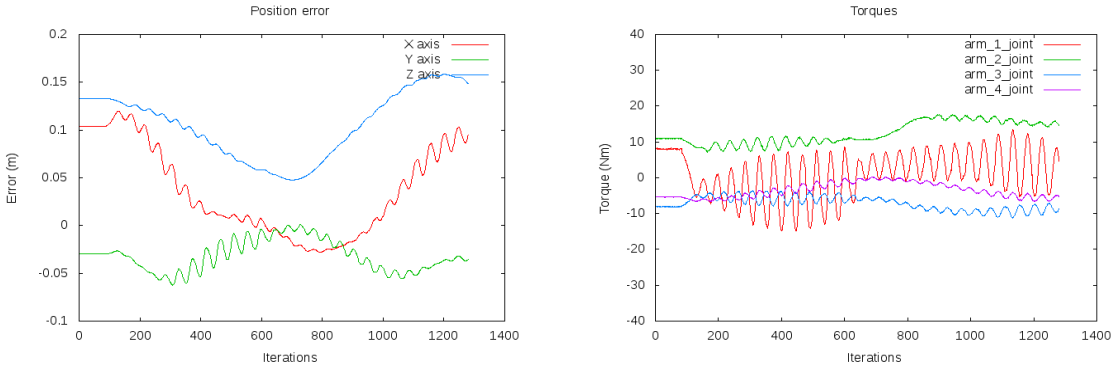


Figura 4.48: Control cartesiano basado en Par-Calculado en robot real. $K_p = diag(400, 400, 400, 200, 200, 200), K_v = 30\boldsymbol{I}$

Como se desprende de las figuras anteriores, el resultado obtenido con este controlador de nuevo está lejos de lo conseguido en el entorno simulado, con errores cartesianos en este caso bastante mayores. Esto se debe a los mismos factores indicados para el control Par-Calculado articular, en lo referente al modelo dinámico del robot y a la frecuencia del ciclo de control.

Así pues, los tres casos propuestos son bastante similares en lo que a error en posición se refiere, con picos máximos de error de 15 centímetros en el eje Z, si bien es cierto que en los otros dos ejes, sobre todo en el Y, el error en ningún momento supera los 5 cm. Con esto, el parámetro que nos hace decantarnos por la figura 4.47 como el mejor resultado de los mostrados en esta memoria es el par articular, puesto que en el caso de la figura 4.46, y sobre todo de la 4.48, éste se vuelve demasiado oscilatorio.

En conclusión, los resultados obtenidos con el control cartesiano basado en Par-Calculado distan mucho de lo obtenido en la simulación, debido principalmente a las discrepancias entre el robot real y su modelo dinámico, pieza clave de cualquier control dinámico, y especialmente en el caso de los basados en Par-Calculado.

4.2.2.2. Control óptimo

El objetivo de este apartado consiste en mostrar los resultados obtenidos al aplicar en el robot TIAGo el controlador óptimo cartesiano descrito por la ecuación (3.32).

Así, al igual que se hizo en entorno simulado, se ha optado por dotar a la matriz W del controlador con el valor M^{-1}, dado que como se ha explicado anteriormente es el que habitualmente ofrece los mejores resultados. Con esto, los valores de error en posición y par articular de las pruebas más características de entre las llevadas a cabo para este controlador vienen representados en las figuras 4.49 a 4.51

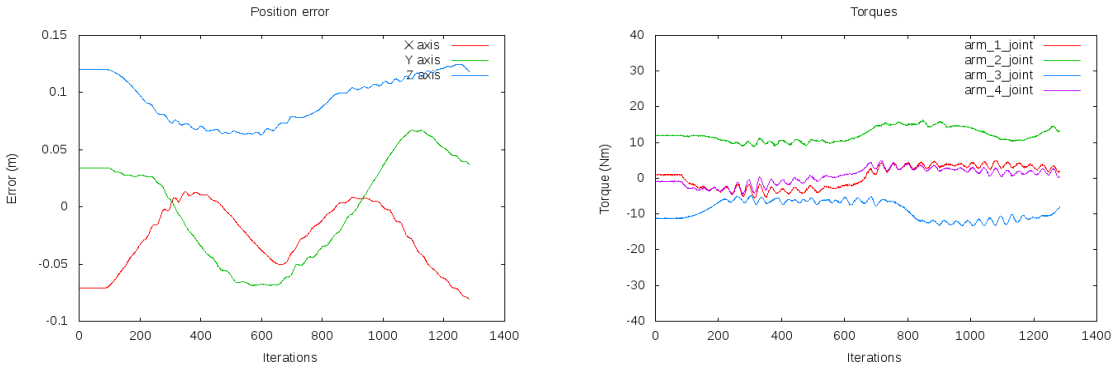


Figura 4.49: Control óptimo cartesiano en robot real. $K_p = diag(100, 100, 100, 120, 120, 120), K_v = 20\boldsymbol{I}$

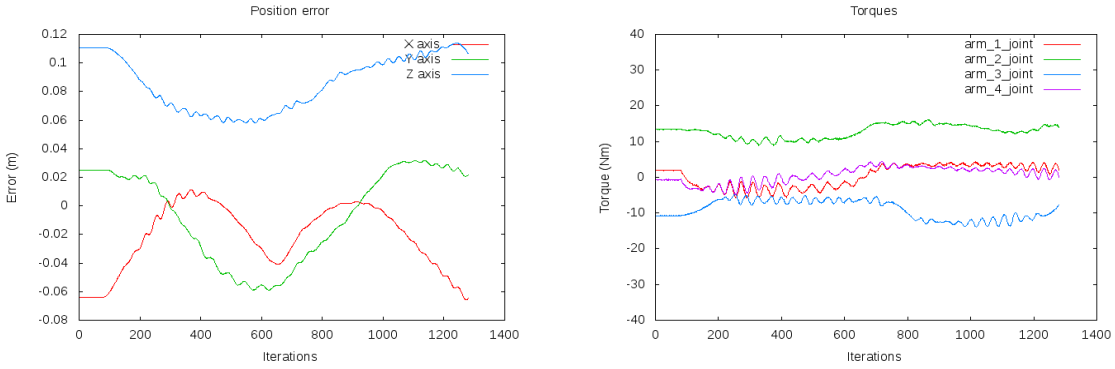


Figura 4.50: Control óptimo cartesiano en robot real. $K_p = diag(150, 150, 150, 120, 120, 120), K_v = diag(35, 35, 35, 20, 20, 20)$

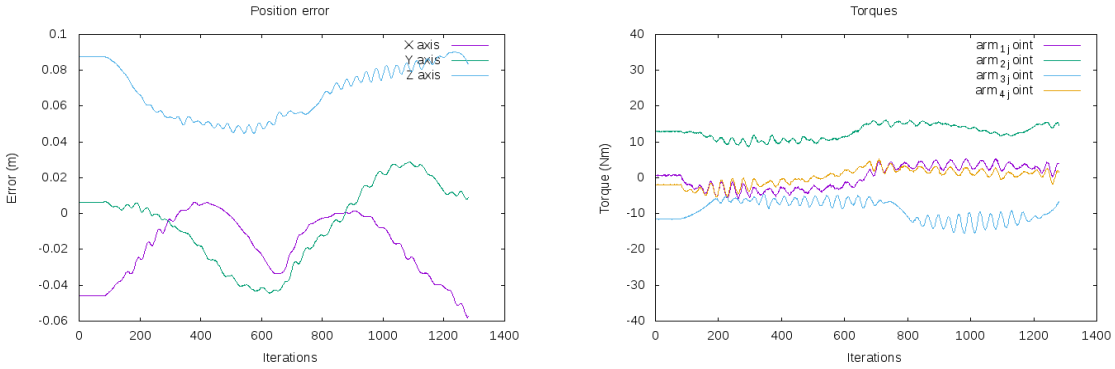


Figura 4.51: Control óptimo cartesiano en robot real. $K_p = diag(200, 200, 200, 150, 150, 150), K_v = diag(40, 40, 40, 25, 25, 25)$

Tal y como se observa en las figuras 4.49 a 4.51, el desempeño del controlador de nuevo está lejos de sus valores en simulación, debido a los mismos factores de los controladores anteriores. Sin embargo, podemos ver como sí que mejora bastante los resultados del control cartesiano basado en Par-Calculado, además de reducir ligeramente el par articular con respecto a dicho controlador, sobre todo en la articulación 1, donde la mejoría es notable.

Así, se puede observar que claramente el mejor resultado se obtiene con la configuración de ganancias PD aplicada en la prueba que se ilustra en la figura 4.51, donde el error medio en los ejes XY es menor de 3 cm, mientras que en el eje Z se tiene un error mayor, de hasta 9 cm, si bien es bastante menor que en el mejor de los casos del control cartesiano basado en Par-Calculado.

En definitiva, a pesar de no obtener unos resultados similares al entorno simulado, en comparativa con el desempeño del control basado en Par-Calculado, el controlador óptimo disminuye notablemente el error cartesiano además de reducir, en mayor o menor medida, los pares articulares ejercidos por los actuadores del robot en la tarea de seguimiento.

4.2.3. Control Visual

En esta última subsección dentro de la experimentación llevada a cabo en el robot real, se va a hacer una reseña a un controlador que no estaba previsto implementar, pero que finalmente se decidió llevar a cabo para el robot real durante la estancia en PAL Robotics.

El controlador en cuestión implementa un control visual directo basado en posición, en el cual se detecta mediante técnicas de visión por computador un marcador *Aruco* [65], del que se extraen sus coordenadas cartesianas y se toman como referencia para un controlador cartesiano. Estos marcadores son similares a los QR estándar, pero se decidió hacer uso de los primeros por el hecho de que en el software incluido en el robot real ya se tenía el paquete de ROS que lleva a cabo el procesamiento de imagen para la detección del marcador, así como la posterior extracción de su posición cartesiana y su publicación mediante un *topic* de ROS. Dicho paquete toma el nombre de *aruco_ros*, y su documentación se puede consultar en [66].

En la figura 4.52 se puede observar un ejemplo de la detección de un marcador de este estilo.

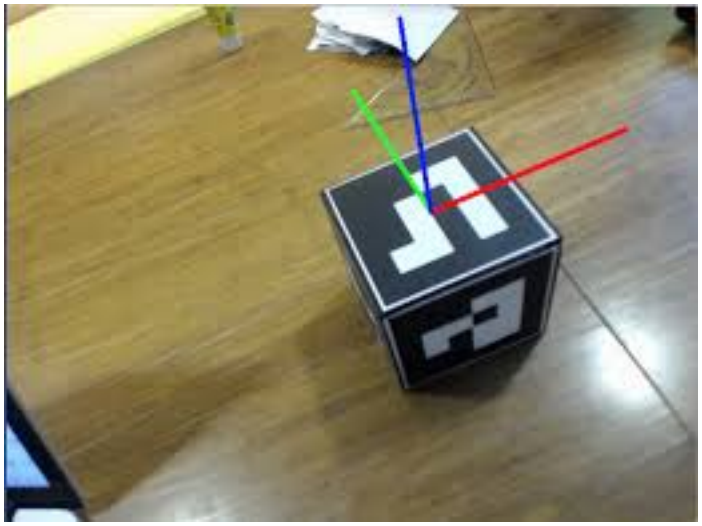


Figura 4.52: Detección de un marcador Aruco. Fuente: `http://www.ahmedahres.com/uploads/7/6/4/7/76477589/bachelor_research_project.pdf`

Como se indicó en el capítulo 3, sección 3.2, el controlador visual óptimo desarrollado en el marco de este proyecto no se pudo implantar en el robot TIAGo al no contar este con una configuración *eye-in-hand*. Por este motivo, se decidió durante la estancia llevada a cabo en PAL Robotics llevar a cabo un control visual basado en posición, ya que lo único que es necesario es, en primer lugar la extracción de la posición cartesiana de la referencia a seguir, cosa que se consigue con el marcador Aruco, y en segundo lugar un control cartesiano, como por ejemplo los indicados en las ecuaciones (3.23) o (3.32),que tome como referencia esta posición del marcador.

Así pues, dado que el único cambio necesario consistía en que el controlador cartesiano leyese del *topic* de ROS correspondiente la posición del marcado Aruco, y usase dicha posición como referencia a seguir, se decidió implementar este comportamiento, pudiéndose observar los resultados obtenidos en la figura 4.53.

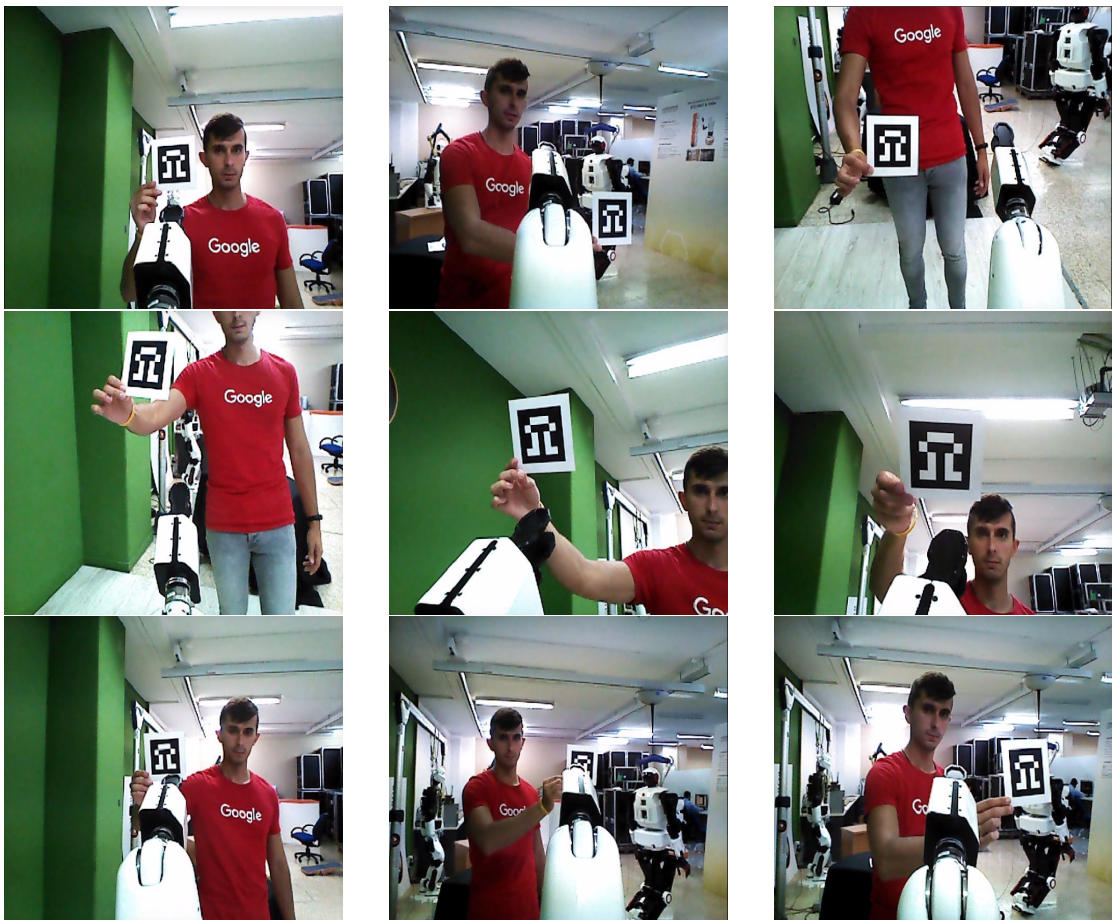


Figura 4.53: Control visual basado en posición. Seguimiento desde la perspectiva de TIAGo.

Tal y como se puede apreciar en la figura 4.53, se lleva a cabo el seguimiento del marcador con el extremo del brazo del robot, dejando una pequeña distancia de seguridad para evitar colisiones con el usuario. Esto se puede ver también desde la perspectiva de un observador externo, donde se aprecia mejor el movimiento del robot conforme se va desplazando el marcador en su entorno.

En la figura 4.54, se puede apreciar con más claridad como varía la posición del brazo al mover el marcador por delante del robot, cumpliendo así con el seguimiento de la referencia de posición cartesiana que el marcador Aruco proporciona al controlador cartesiano.

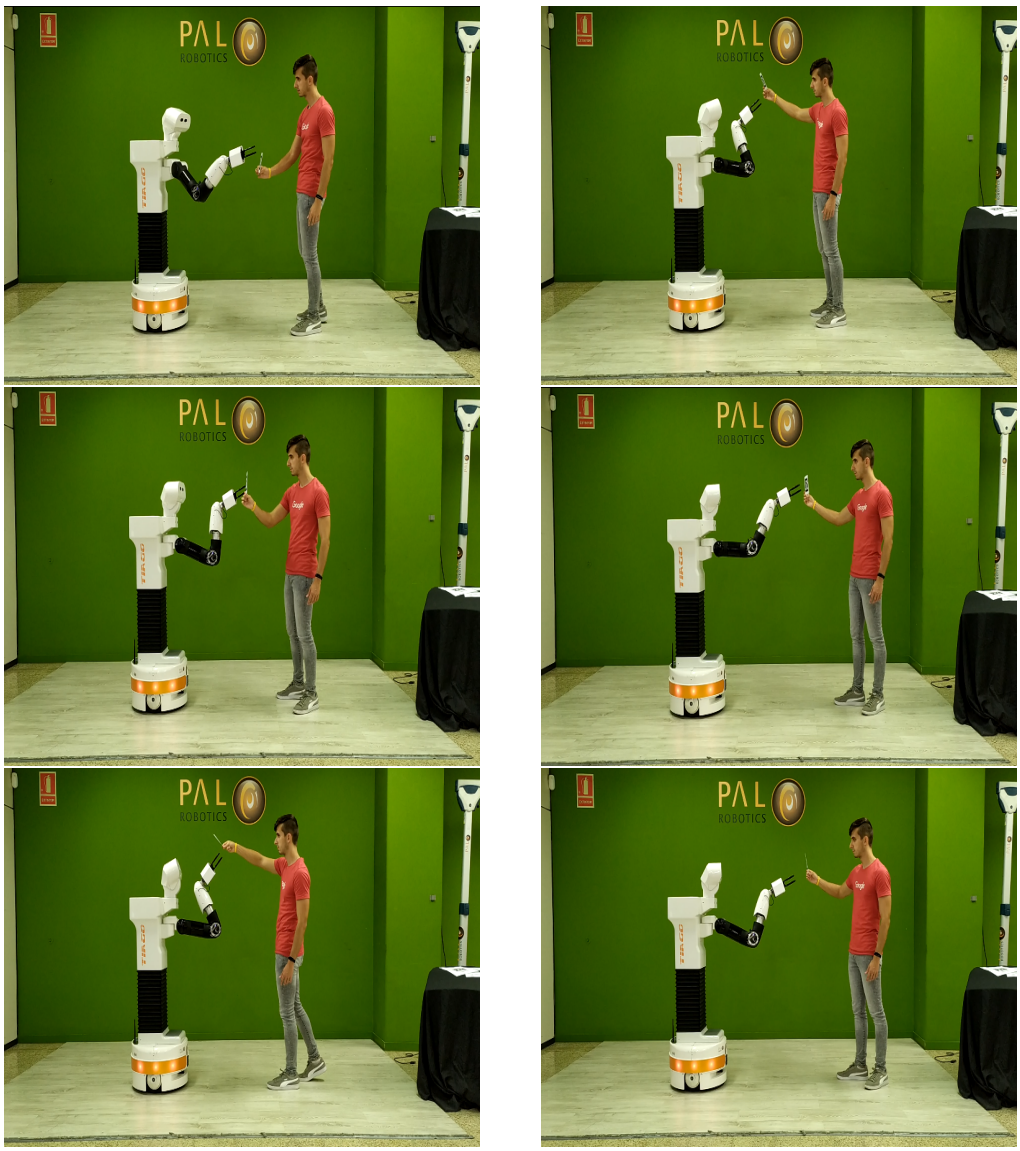


Figura 4.54: Control visual basado en posición. Seguimiento desde la perspectiva de un observador externo.

Así pues, se puede comprobar el correcto funcionamiento del controlador visual basado en posición implementado en el marco de este proyecto, donde la parte de extracción de la posición en el espacio cartesiano del marcador la realiza íntegramente el paquete de ROS correspondiente, teniendo únicamente que adaptar la referencia del controlador cartesiano para que en lugar de cogerla del planificador de trayectorias, la tome del *topic* publicado por dicho paquete.

Con esto, concluye la exposición de los resultados más representativos obtenidos durante la fase de experimentación de este proyecto, y se va a dar paso a la última sección de este capítulo, en la que se va a analizar y comparar dichos resultados, con el objetivo de entender por qué se han dado así.

4.3. Análisis y comparativa de resultados

En esta última sección, se va a proceder a la comparativa de resultados entre los distintos controladores implementados en el marco de este proyecto, así como a analizar y justificar las diferencias entre el comportamiento de los controladores en entorno simulado y aplicados al robot real.

Para ello, en primer lugar se van a comparar entre ellos los controladores articulares en entorno simulado, seguidos de los controladores en espacio cartesiano, para así observar cual ofrece el mejor comportamiento en condiciones ideales. Tras esto, se repetirá el mismo proceso aplicándolo a la experimentación llevada a cabo en el robot real.

Finalmente, se analizarán algunos de los posibles motivos de la discrepancia observada entre los resultados en entorno simulado y los obtenidos en las pruebas realizadas en el robot real, tratando así de justificar el por qué de esta diferencia, y del mismo modo encontrar posibles soluciones al respecto.

Así pues, en base a los resultados obtenidos en la fase de simulación, en lo que respecta a los controladores articulares el comportamiento desempeñado por todos ellos es muy preciso, con errores inferiores a los 0.005 *rad.* en todos los casos, así como un valor adecuado de los pares articulares, lejos de la saturación en los casos en que no se producía oscilación. Además, se aprecia una gran similitud entre los controladores PD con prealimentación y PD+, lo cual no es una sorpresa dado que sus leyes de control son casi idénticas.

Con esto, el controlador que mejores resultados arroja es el control Par-Calculado, el cual en su mejor configuración tiene un error máximo de 0.001 *rad.*, y un error medio de ± 0.0002 *rad.* Esto se debe a que, en condiciones ideales, con el control Par-Calculado se consigue un comportamiento lineal en bucle cerrado, tal y como se explica en el capítulo 3. Sin embargo, el hecho de otorgar un papel tan protagonista al modelo dinámico implica que si éste no es demasiado bueno, se pierda este buen comportamiento del controlador. Aun así, estos resultados en entorno simulado nos ofrecen una comparativa válida para los casos prácticos en los que se puedan llegar a conseguir condiciones cercanas a las ideales.

En la tabla 4.1, se muestra un resumen con los resultados de cada uno de los controladores articulares con su mejor configuración de ganancias PD.

Controlador	**Error máx. (rad)**	**Error medio (rad)**	**Valor de K_p**	**Valor de K_v**
PD con prealimentación	0.004	0.002	$1000\boldsymbol{I}$	$100\boldsymbol{I}$
PD+	0.004	0.002	$1000\boldsymbol{I}$	$100\boldsymbol{I}$
Par-Calculado	0.001	0.0002	$50000\boldsymbol{I}$	$500\boldsymbol{I}$

Tabla 4.1: Mejores resultados de controladores articulares en simulación.

Por otro lado, en lo que a los controladores cartesianos se refiere, de nuevo se obtienen buenos resultados, consiguiendo altos niveles de precisión tanto con el control cartesiano basado en Par-Calculado como con el control óptimo cartesiano.

En ambos casos, los resultados obtenidos son muy similares, obteniendo errores máximos muy cercanos al milímetro en el eje Z, mientras que en los ejes XY se oscila entre valores muy bajos, de entre los 0.2 y los 0.5 milímetros, consiguiendo así una precisión muy elevada.

Así pues, en lo referente a error en posición apenas hay variación entre los dos controladores, ya que el basado en Par-Calculado obtiene mejor comportamiento para los ejes XY, mientras que el control óptimo reduce el error en Z, si bien es cierto que en ambos casos la diferencia es ínfima.

En la tabla 4.2, se muestra un resumen con los resultados de cada uno de los controladores cartesianos en el caso de sus mejores configuraciones de ganancias PD.

Controlador	**Error máx. (m)**	**Error medio (m)**	**Valor de K_p**	**Valor de K_v**
Basado en Par-Calculado	0.001	0.0005	$20000\boldsymbol{I}$	$300\boldsymbol{I}$
Óptimo	0.0009	0.0005	$20000\boldsymbol{I}$	$100\boldsymbol{I}$

Tabla 4.2: Mejores resultados de controladores cartesianos en simulación.

Una vez analizados los resultados obtenidos en entorno simulado, a continuación se va a mostrar la comparativa entre los resultados de las pruebas llevadas a cabo con el robot real.

En general, se ha observado en este capítulo como el comportamiento de los controladores implementados en el robot real en ocasiones han distado demasiado de los resultados obtenidos en el entorno simulado. Es evidente que en ningún caso al llevar a una plataforma física real un controlador se van a conseguir exactamente los mismos resultados que en simulación, ya que siempre van a aparecer perturbaciones, imperfecciones y demás que van a afectar al funcionamiento del sistema. Sin embargo, a este hecho común en cualquier salto desde un entorno ideal a uno real, se le han unido otros factores consecuencia del hardware y software propio del robot TIAGo, que han provocado que los resultados obtenidos no sean tan correctos.

Uno de los factores más determinantes ha sido la frecuencia de actualización de la acción de control en el sistema de tiempo real interno del robot. Normalmente, en los casos en los que se emplea un control directo de los actuadores de un robot manipulador, el ciclo de control se suele ejecutar a una frecuencia de 1000 Hz, actualizando por tanto la acción de control cada milisegundo, lo cual sucede en el caso del robot simulado, donde los resultados son considerablemente buenos. Sin embargo, en la arquitectura software del robot real el ciclo de control se ejecutaba con una frecuencia de 100 Hz, diez veces más lento que lo habitual, lo cual hace que el control directo se vuelva oscilante e impreciso al no actualizar las acciones de control a la velocidad necesaria.

Por otro lado, el otro factor clave que explica la diferencia de resultados entre la simulación y el robot real es la fidelidad del modelo dinámico del robot. En el entorno simulado, el modelo dinámico del robot representa exactamente la dinámica de este en el simulador, puesto que el modelo del robot en Gazebo se genera directamente del URDF del TIAGo, que es de donde KDL obtiene los parámetros dinámicos del modelo que se emplea en los controladores. Sin embargo, en el robot real el modelo dinámico generado por el URDF no tiene en cuenta gran parte de las propiedades dinámicas de los actuadores del robot, como por ejemplo rozamientos, elasticidad de las reductoras, transmisión de par entre motor y articulación, etc.

Además, como se ha comentado en simulación sí que se lleva a cabo un control directo del par articular ejercido por las articulaciones, mientras que en el robot real solo se permite un control por corriente, siendo necesaria la transformación de la acción de control multiplicando por una serie de constantes dependientes de la fabricación del motor, lo cual introduce una nueva posible fuente de error en el sistema.

En definitiva, todos los factores anteriores contribuyen a que los resultados obtenidos en el robot real no sean tan correctos como debiesen. Aun así, se ha conseguido poner en funcionamiento los controladores implementados en un robot real de alta complejidad como es el TIAGo, y, aunque los resultados no sean los ideales, tampoco son en absoluto negativos en la mayoría de casos.

Sin más, a continuación se va a exponer la comparativa entre los resultados obtenidos en las pruebas realizadas en el robot real.

Empezando por los controladores articulares, el comportamiento desempeñado por los controladores PD con prealimentación y PD+ es bastante aceptable teniendo en cuenta los problemas que hemos mencionado en el robot real, debido a que el control PD compensa en cierta medida el error introducido por la presencia de un modelo dinámico poco exhaustivo, así como la baja frecuencia del ciclo de control.

Sin embargo, el control Par-Calculado no ofrece resultados aceptables, debido a la dependencia de este controlador del modelo dinámico, que como se ha dicho no es demasiado preciso. Este hecho unido a la frecuencia baja del controlador tiene como consecuencia los resultados obtenidos.

Con esto, el controlador que mejores resultados arroja es el control PD+, el cual en su mejor configuración tiene un error máximo de 0.08 *rad.*, y un error medio de ± 0.04 *rad.*Esto como se ha indicado, el hecho de que el el control PD no se vea afectado por el modelo dinámico es el motivo principal del comportamiento más correcto de este controlador, unido a que se considera le modelo dinámico en el estado actual del robot en lugar del deseado, como sucede en el control PD con prealimentación, en el que los resultados son ligeramente peores.

En la tabla 4.3, se muestra un resumen con los resultados de cada uno de los controladores articulares aplicados al robot real con su mejor configuración de ganancias PD.

Controlador	**Error máx. (rad)**	**Error medio (rad)**	**Valor de K_p**	**Valor de K_v**
PD con prealimentación	0.085	0.04	$100\boldsymbol{I}$	$30\boldsymbol{I}$
PD+	0.075	0.04	$diag(100, 100, 120, 100)$	$diag(20, 20, 30, 20)$
Par-Calculado	0.3	0.015	$diag(100, 200, 400, 200)$	$diag(20, 28, 35, 20)$

Tabla 4.3: Mejores resultados de controladores articulares en robot real.

Por su parte, en lo que a los controladores cartesianos se refiere, los resultados obtenidos de nuevo vuelven a empeorar lo que se tuvo en la simulación. Sin embargo, los resultados obtenidos no son del todo malos, dado que el TIAGo no es un robot destinado al trabajo de precisión, por lo que los errores cartesianos son asumibles en el caso de los ejes XY, siendo el Z el más problemático de todos.

Así pues,en base a la experimentación realizada, se puede concluir que el mejor desempeño viene dado por el control óptimo, lo cual se podía intuir según el desarrollo teórico expuesto en el capítulo 3, pero que hasta no ser comprobado con datos empíricos no se puede asegurar.

Además, si bien es cierto que la mejoría debiera ser mayor, en la experimentación sí que se aprecian los dos efectos esperados del control óptimo, como son la reducción de error en el seguimiento de la trayectoria, así como la optimización de los pares articulares.

En la tabla 4.4, se muestra un resumen con los resultados tomados del robot real de cada uno de los controladores cartesianos en el caso de sus mejores configuraciones de ganancias PD.

Controlador	**Error máx. (m)**	**Error medio (m)**	**Valor de K_p**	**Valor de K_v**
Basado en Par-Calculado	0.15	0.05	$\begin{pmatrix} 300\boldsymbol{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & 150\boldsymbol{I}_3 \end{pmatrix}$	$20\boldsymbol{I}$
Óptimo	0.09	0.03	$\begin{pmatrix} 200\boldsymbol{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & 150\boldsymbol{I}_3 \end{pmatrix}$	$\begin{pmatrix} 40\boldsymbol{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & 25\boldsymbol{I}_3 \end{pmatrix}$

Tabla 4.4: Mejores resultados de controladores cartesianos en robot real.

Con esto, termina este cuarto capítulo, donde se han presentado, comparado, analizado y justificado los resultados obtenidos durante la fase de experimentación de los controladores diseñados e implementados en el capítulo 3, tanto en entorno simulado como con el robot TIAGo en la sede de PAL Robotics.

5. Conclusiones

En este quinto y último capítulo, una vez desarrollada al completo la memoria, donde se ha expuesto el interés de la problemática abordada, los trabajos más destacados en similares lineas de investigación, el desarrollo teórico y la implementación de los controladores, y por último los resultados obtenidos tras su implantación, se puede afirmar que se han cumplido los objetivos planteados al inicio de este proyecto.

En primer lugar, se ha llevado a cabo una reseña histórica donde se muestra el interés del ser humano de "dar vida" a sus creaciones, lo cual con el paso de los años dio lugar a los primeros automatismos, para acabar desembocando en el concepto de robótica que se tiene hoy en día. Así pues, partiendo de supuestos mitológicos y de ciencia-ficción, se ha convergido a la estructura general de cualquier robot, y destacado la importancia del sistema de control como elemento conector entre los sistemas de percepción y los actuadores del robot.

Tras esto, se han mencionado algunos de los trabajos más destacados relacionados con el control de robots de forma genérica, diferenciando control cinemático de control dinámico, y profundizando más en este segundo, concretamente en lo que a control dinámico multiarticular de robots manipuladores se refiere, al ser el tema principal de este proyecto. Además, se ha expuesto también el estado del arte en control óptimo de cualquier tipo de robots, destacando las ventajas que éste control tiene en lo que a precisión en el seguimiento y reducción de pares articulares necesarios se refiere. Finalmente, se concluyó el estado del arte con la explicación de los distintos tipos de control visual existentes, así como la mención de algunos de los trabajos más destacados en este campo.

Una vez puesta en contexto la temática del trabajo gracias al análisis del estado del arte en

los campos mencionados, en el capítulo 3 se ha llevado a cabo en primer lugar el desarrollo teórico de diversos controladores dinámicos multiarticulares, de los cuales se ha expuesto su ley de control y se han analizado sus equilibrios y su estabilidad asintótica matemáticamente. De entre estos controladores, destaca el control óptimo de robots manipuladores, así como su extensión a control visual directo, con los controladores diseñados en el marco de este proyecto.

Así, tras esta parte de análisis y desarrollo teórico, se ha pasado a la fase de implementación, en donde se ha puesto en conocimiento del lector las características hardware y software del robot TIAGo, escogido para la implantación de los controladores diseñados, así como la implementación de éstos en ROS Control y su incorporación a la arquitectura software del robot.

Finalmente, en el cuarto capítulo se han presentado los resultados obtenidos durante la fase de experimentación de este proyecto, llevada a cabo en primer lugar mediante una simulación dinámica en Gazebo, y posteriormente aplicando los controladores implementados al robot TIAGo real en la sede de PAL Robotics.

Así pues, tras realizar el análisis de los resultados, observamos como en entorno simulado éstos son considerablemente buenos, con errores de seguimiento inferiores al control PID en posición por defecto del robot en el caso del control articular, reduciendo además las grandes oscilaciones causadas por éste. Por su parte, los controladores cartesianos también consiguen un desempeño notable, con errores en el seguimiento del orden del milímetro.

Por otro lado, en lo que a las pruebas en el robot real respecta, se ha observado como en general los resultados obtenidos no son tan buenos como se esperaba tras la fase de simulación, debido principalmente a que la frecuencia del ciclo de control en el robot real es de 100 Hz, muy baja para llevar a cabo un control directo como el propuesto en este proyecto, lo cual dificulta el seguimiento de la trayectoria e introduce error y oscilaciones. Además, en el caso del robot real el modelo dinámico representado en el URDF es un modelo genérico que no representa las características del robot concreto con el que se trabajó, distando mucho de

la realidad, lo cual conlleva que los controladores dinámicos pierdan parte de su eficacia.

Sin embargo, aun con estos problemas la conclusiones extraídas de esta experimentación son positivas, puesto que pese a la lenta frecuencia del ciclo de control y las diferencias entre el modelo dinámico del robot, los controladores demuestran su robustez y hacen que se siga la trayectoria de todos modos. Además, el problema relativo al modelo dinámico se podría solucionar si se obtuviese el modelo dinámico del robot en concreto que se esté utilizando, conociendo al detalle el hardware que lo compone.

5.1. Artículos relacionados

A lo largo del desarrollo de este proyecto, se han llevado a cabo distintas investigaciones en lo referente al diseño de controladores dinámicos multiarticulares óptimos, de entre las cuales han surgido diversas publicaciones, que pueden servir para extender el trabajo que se ha presentado en esta memoria.

En primer lugar, se publicó en las Jornadas de Automática del Comité Español de Automática (CEA), las cuales tuvieron lugar del 5 al 7 de septiembre de 2018 en Badajoz, el artículo *Optimización en Control Visual de Robots Manipuladores* [32]. Este artículo presenta el controlador visual óptimo basado en imagen desarrollado en el capítulo 3, sección 3.2.2, mostrando además resultados en simulación que verifican los efectos esperados del controlador.

Cabe destacar que, gracias a esta investigación, se recibió el *Premio a Mejor Trabajo en Robótica 2018* por parte del CEA y patrocinado por la empresa Robotnik, lo cual es una garantía de que el trabajo realizado genera gran interés entre la comunidad científica, siendo reconocido por uno de los organismos más importantes a nivel nacional en el campo de la robótica.

En segundo lugar, se extendió el controlador visual óptimo anterior para el guiado de robots manipuladores móviles, como es por ejemplo el robot TIAGo, consiguiendo aunar en

un único controlador la movilidad del brazo manipulador y de la base móvil de robots de estas características. Esta investigación se publicó en la revista científica *Electronics* el 27 de Marzo de 2019 mediante un artículo de nombre *Optimal Image-Based Guidance of Mobile Manipulators using Direct Visual Servoing* [33], en el cual se presenta el desarrollo teórico del controlador en cuestión, así como los resultados de la experimentación en entorno simulado que validan el diseño de éste.

Finalmente, una extensión del caso anterior ha sido presentada para las próximas *Jornadas de Automática del CEA 2019*, publicándose esta investigación bajo el título *Control Dinámico de Manipuladores Móviles con Realimentación Visual*.

Así pues, se puede observar como el alcance de la investigación realizada en el marco de este proyecto va más allá de lo puramente presentado en esta memoria, así como el interés que despierta esta investigación tanto a la comunidad científica como a la empresa privada, lo cual es un aval y una motivación adicional de cara a seguir trabajando en esta línea.

5.2. Trabajos futuros

Así pues, una vez expuestas las conclusiones alcanzadas tras el desarrollo del proyecto en su totalidad, así como presentados los artículos derivados de la realización de éste proyecto, tan solo resta indicar brevemente algunos de los posibles trabajos futuros a partir de este proyecto.

A nivel teórico, resultaría interesante incluir en el controlador óptimo un término de control de fuerza e impedancia, que permitiese controlar de forma precisa y segura la interacción del robot con el entorno, ya sea para tareas de manipulación de objetos, o incluso para la interacción con seres humanos propia de robots de servicio colaborativos como TIAGo entre otros.

Por otro lado, a nivel práctico, debido a los problemas con el hardware que se han encontrado para la implementación de los controladores diseñados en este proyecto, un posible trabajo sería la extensión y desarrollo de los controladores para robots con hardware de alta gama, como por ejemplo el robot *Talos* de PAL Robotics, que cuenta con la posibilidad de

realizar control directo de todas sus articulaciones, sensores de par articular, y una frecuencia de ejecución del ciclo de control de 1000 Hz, diez veces mayor que TIAGo, lo cual permitiría un correcto control dinámico de las articulaciones del robot.

Bibliografía

[1] J. Pomares, G.J. García, J. Pérez, P. Gil, and F. Torres. *Control Visual. Conceptos y métodos en visión por computador.* CEA, 2016.

[2] Rafael Kelly and Víctor Santibáñez. *Control de Movimiento de Robots Manipuladores.* Pearson Prentice-Hall, 2003.

[3] PAL Robotics S.L. *TIAGo Handbook*, 2019.

[4] Carlos García Gual. *Prometeo: Mito y literatura.* Fondo de cultura económica de España, 2009.

[5] Mary Shelley. *Frankenstein; or, The Modern Prometheus.* Lackington, Hughes, Harding, Mavor & Jones, Gradifco, 1818.

[6] William Smith. *A Dictionary of Greek and Roman Biography and Mythology.* Boston: Little, Brown & Co, 1849.

[7] Karel Čapek. *RUR. Robots Universales Rossum : obra en tres actos y un epílogo.* Barcelona: Círculo de lectores, 2004.

[8] Isaac Asimov. *The Complete Robot.* Barcelona: Martínez Roca, 1984.

[9] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control.* Springer-Verlag, 2010.

[10] Antonio Barrientos, Luis F. Peñín, Carlos Balaguer, and Rafael. Aracil. *Fundamentos de Robótica.* McGraw-Hill, 2007.

[11] K.S Fu, R.C Gonzalez, and C.S.G Lee. *Planificación de Trayectorias de un Manipulador.* McGraw-Hill, 1988.

[12] Alessandro De Luca. Dynamic model of robots: Newton-euler approach. `http://www.diag.uniroma1.it/~deluca/rob2_en/06_NewtonEulerDynamics.pdf`.

[13] John J. Craig. *Introduction to robotics: Mechanics and Control*. Addison-Wesley, 1989.

[14] Foudil Abdessemed. Svm-based control system for a robot manipulator. *International Journal of Advanced Robotic Systems*, 9:1, 12 2012.

[15] Bruno Siciliano and Oussama Khatib. *Handbook of robotics*. Springer-Verlag, 2008.

[16] O. Koç, G. Maeda, and J. Peters. Optimizing the execution of dynamic robot movements with learning control. *IEEE Transactions on Robotics*, 2018.

[17] K. Al-Khudir, G. Halvorsen, L. Lanari, and A. De Luca. Stable torque optimization for redundant robots using a short preview. *IEEE Robotics and Automation Letters*, 4(2):2046–2053, 2019.

[18] J. Luo, Y. Zhao, D. Kim, O. Khatib, and L. Sentis. Locomotion control of three dimensional passive-foot biped robot based on whole body operational space framework. In *Proceedings of the IEEE International Conference on Robotics and Biometrics (ROBIO)*, page 1577, 2017.

[19] G. Antonelli, E. Cataldi, F. Arrichiello, P. Robuffo Giordano, S. Chiaverini, and A. Franchi. Adaptive trajectory tracking for quadrotor mavs in presence of parameter uncertainties and external disturbances. *IEEE Transactions on Control Systems Technology*, 26(0):248–254, 2018.

[20] L. Andolfatto, S. Lavernhe, and J.R.R Mayer. Evaluation of servo, geometric and dynamic error sources on five-axis high-speed machine tool. *International Journal of Machine Tools & Manufacture*, 51:787–796, 2011.

[21] Oussama Khatib. Dynamic control of manipulators in operational space. *Proceedings of the 6th IFTOMM Congress on Theory of Machines and Mechanisms*, pages 1–10, 1983.

[22] B. Xian, M. Queiroz, D. Dawson, and I. Walker. Task space tracking control of redundant robot manipulators via quaternion feedback. *Proceedings of IEEE International Conference on Control Applications*, pages 363–368, 2001.

[23] E. Zengeroglu, D. Dawson, I. Walker, and A. Behl. Nonlinear tracking control of kinematically redundant robot manipulators. *Proceedings of the American Control Conference*, pages 2513–2517, 2000.

[24] F.E. Udwadia. A new perspective on tracking control of nonlinear structural and mechanical systems. In *Proceedings of the Royal Society of London Series A*, pages 1783–1800, 2003.

[25] H. Bruyninckx and O. Khatib. Gauss' principle and the dynamics of redundant and constrained manipulators. In *Proceedings of the 2000 IEEE international conference on robotics & automation*, pages 2563–2569, 2000.

[26] F.E. Udwadia and R.E. Kalaba. Analytical dynamics: a new approach. *Cambridge University Press*, 2003.

[27] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Operational space control: A theoretical and empirical comparision. *International Journal of Robotics Research*, 27(6):737–757, 2008.

[28] N. Kumar, J. Borm, V. Panwar, and J. Chai. Tracking control of redundant robot manipulators using rbf neural network and an adaptive bound on disturbances. *International Journal of Precision Engineering and Manufacturing*, 13(8):1377–1386, 2012.

[29] Young H. Kim, Frank L. Lewis, and Darren M. Dawson. Intelligent optimal control of robotic manipulators using neural networks. *Automatica*, 36(9):1355 – 1364, 2000.

[30] Gerasimos Rigatos, Krishna Busawon, Jorge Pomares, Patrice Wira, and Masoud Abbaszadeh. A nonlinear optimal control approach for the spherical robot. pages 2496–2501, 2018.

[31] G. Rigatos, K. Busawon, J. Pomares, and M. Abbaszadeh. Nonlinear optimal control for the wheeled inverted pendulum system. *Robotica*, page 1–19, 2019.

[32] Álvaro Belmonte, Jorge Pomares, and Gabriel Jesús García. Optimización en control visual de robots manipuladores. In *XXXIX Jornadas de Automática del CEA*, page 1109, 2018.

[33] Álvaro Belmonte, José L. Ramón, Jorge Pomares, Gabriel J. Garcia, and Carlos A. Jara. Optimal image-based guidance of mobile manipulators using direct visual servoing. *Electronics*, 8(4), 2019.

[34] J. Hill and W. Park. Real time control of a robot with a mobile camera. In *Proceedings of the 9th International Symposium on Industrial Robots*, pages 233–246, 1979.

[35] A C. Sanderson and L E. Weiss. Image-based visual servo control using relational graph error signals. *Proc. IEEE*, 1980.

[36] F. Chaumette and S. Hutchinson. Visual servo control. part 1: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, 2006.

[37] F. Chaumette and S. Hutchinson. Visual servo control. part 2: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007.

[38] Graziano Chesi and Koichi Hashimoto. *Visual Servoing via Advanced Numerical Methods.* Springer-Verlag, 2010.

[39] P. K. Allen, B. Yoshimi, and A. Timcenko. Real-time visual servoing. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, volume 1, pages 851–856, 1991.

[40] D.F. Dementhon and L.S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2):123–141, 1995.

[41] D. Oberkampf, D.F. Dementhon, and L.S. Davis. Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63(3):495–511, 1996.

[42] E. Cervera, F. Berry, and P. Martinet. Stereo visual servoing with a single point: A comparative study. In *Proceedings of the IEEE International Conference on Advanced Robotics*, pages 213–218, 2001.

[43] Rafael Kelly, Ilse Cervantes, Jose Alvarez-Ramirez, E Bugarin, and Carmen Monroy. *On Transpose Jacobian Control for Monocular Fixed-Camera 3D Direct Visual Servoing.* 09 2008.

[44] J. Pomares, J.A. Corrales, G.J. García, and F. Torres. Direct visual servoing to track trajectories in human-robot cooperation. *International Journal of Advanced Robotics Systems*, 8(4):44, 2011.

[45] C.C. Cheah, C. Liu, and J.J.E. Slotine. Adaptive jacobian vision based control for robots with uncertain depth information. *Automatica*, 46(7):1228–1233, 2010.

[46] Z. Zake, F. Chaumette, N. Pedemonte, and S. Caro. Vision-based control and stability analysis of a cable-driven parallel robot. *IEEE Robotics and Automation Letters*, 4(2):1029–1036, 2019.

[47] B. Penin, P. R. Giordano, and F. Chaumette. Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE Robotics and Automation Letters*, 3(4):3725–3732, 2018.

[48] M. Spong and M. Vidyasagar. *Robot dynamics and control*. John Wiley & Sons, 1989.

[49] P. Khosla and T. Kanade. Parameter identification of robot dynamics. In *Proceedings of the 24th IEEE Conference on Decision and Control*, 1985.

[50] M. Vidyasagar. *Nonlinear systems analysis*. Prentice-Hall, 1993.

[51] S. Sastry. *Nonlinear systems: Analysis, Stability and Control*. Springer-Verlag, 1999.

[52] Ubuntu 16.04 reference. `http://releases.ubuntu.com/16.04/`.

[53] Xenomai real-time patch. `https://xenomai.org/`.

[54] The orocos project. `http://www.orocos.org/`.

[55] Robot operative system (ros). `https://www.ros.org/`.

[56] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo. ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*, 2017.

[57] The construct: Robot ignite academy. ros control course. https://www.robotigniteacademy.com/en/course/ros-control-101_12_0/.

[58] Gazebo simulator. http://gazebosim.org/.

[59] Wiki oficial del robot tiago. http://wiki.ros.org/Robots/TIAGo.

[60] Wiki oficial del paquete urdf. http://wiki.ros.org/urdf.

[61] Kinematics and dynamics library (kdl) documentation. http://docs.ros.org/kinetic/api/orocos_kdl/html/index.html.

[62] Eigen: C++ library for algebraic calculus. documentation. http://eigen.tuxfamily.org/index.php?title=Main_Page.

[63] Dynamic reconfigure package. documentation. http://wiki.ros.org/dynamic_reconfigure.

[64] Universidad de Sevilla. Principio de d'alembert. http://laplace.us.es/wiki/index.php/Principio_de_D%27Alembert_(CMR).

[65] Francisco Romero Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 2018.

[66] Ros aruco package documentation. http://wiki.ros.org/aruco_ros.

Lista de Acrónimos y Abreviaturas

CEA	Comité Español de Automática.
GDL	grados de libertad.
HMI	Interacción Humano - Máquina.
IA	Inteligencia Artificial.
KDL	The Kinematics and Dynamics Library.
PD	Proporcional-Derivativo.
ROS	Robot Operative System.
SLAM	Localización y mapeo simultáneo.
TFG	Trabajo Final de Grado.
TFM	Trabajo Final de Máster.
URDF	Unified Robot Description Format.

A. Anexo I: Código de los controladores dinámicos multiarticulares

Código A.1: Código C++ de los controladores dinámicos multiarticulares

```
1
2 #ifndef DIRECT_DYNAMIC_CONTROLLER_HARDWARE_INTERFACE_ADAPTER_H
3 #define DIRECT_DYNAMIC_CONTROLLER_HARDWARE_INTERFACE_ADAPTER_H
4
5 // C++ Standard headers
6 #include <cassert>
7 #include <string>
8 #include <vector>
9 #include <iostream>
10
11 // Boost headers
12 #include <boost/shared_ptr.hpp>
13
14 // ROS headers
15 #include "ros/ros.h"
16 #include <ros/node_handle.h>
17 #include <ros/time.h>
18 #include "std_msgs/Float64MultiArray.h"
19 #include <ddynamic_reconfigure/ddynamic_reconfigure.h>
20
21 // ROS Control headers
22 #include <control_toolbox/pid.h>
23 #include <hardware_interface/joint_command_interface.h>
24 #include <hardware_interface/posvel_command_interface.h>
25 #include <hardware_interface/posvelacc_command_interface.h>
26
27 // Eigen
28 #include <Eigen/Dense>
29 #include <unsupported/Eigen/MatrixFunctions> // To get sqrt of a matrix
```

```
#include <Eigen/Geometry>

// KDL includes
#include <kdl_parser/kdl_parser.hpp>
#include <urdf/model.h>
#include <kdl/chaindynparam.hpp>
#include <kdl/chain.hpp>
#include <kdl/chainfksolverpos_recursive.hpp>
#include <kdl/frames.hpp>
#include <kdl/framevel.hpp>
#include <kdl/jacobian.hpp>
#include <kdl/chainjnttojacsolver.hpp>
#include <kdl/jntarray.hpp>
#include <kdl/jntarrayvel.hpp>
#include <kdl/jntarrayacc.hpp>


template <class HardwareInterface, class State>
class HardwareInterfaceAdapter
{
public:
  bool init(std::vector<typename HardwareInterface::ResourceHandleType>& /*joint_handles*/, ros::↩
      ↪ NodeHandle& /*controller_nh*/)
  {
    return false;
  }

  void starting(const ros::Time& /*time*/) {}
  void stopping(const ros::Time& /*time*/) {}

  void updateCommand(const ros::Time& /*time*/,
                     const ros::Duration& /*period*/,
                     const State& /*desired_state*/,
                     const State& /*state_error*/) {}
};

/**
 * \brief Adapter for an effort-controlled hardware interface. Maps position and velocity errors to effort ↩
     ↪ commands
 * through diferent strategies, which can be chosen at the YAML config file. These are:
 * - Feedforward
 * - PD_Feedforward
 * - PD+
```

```
71 * - Computed_torque
72 * - Cartesian_CT
73 * - Optimal
74 */
75 template <class State>
76 class HardwareInterfaceAdapter<hardware_interface::EffortJointInterface, State>
77 {
78 public:
79   HardwareInterfaceAdapter() : joint_handles_ptr_(0) {}
80
81   bool init(std::vector<hardware_interface::JointHandle>& joint_handles, ros::NodeHandle& controller_nh)
82   {
83     // Store pointer to joint handles
84     joint_handles_ptr_ = &joint_handles;
85     controller_nh_ptr_ = &controller_nh;
86
87     // Get joint names from parameter server
88     if (!controller_nh_ptr_->getParam("joints", joint_names_)){
89       ROS_ERROR("Could not load joint names from parameter server");
90     }
91
92     // Obtain proportional and derivative gains stated at the YAML config file
93     double kp_default, kv_default = 0.0;
94
95     if (!controller_nh_ptr_->getParam("Kp", kp_default)){
96       ROS_ERROR("Could not load proportional gain. Default value Kp=500");
97     }
98
99     if (!controller_nh_ptr_->getParam("Kv", kv_default)){
100      ROS_ERROR("Could not load derivative gain. Default value Kv=50");
101    }
102
103     // Obtain required controller type. Default: PD+
104    if (!controller_nh_ptr_->getParam("controller_type", controller_type)){
105      ROS_ERROR("Could not load controller type. Default: PD+");
106    }
107
108    // Friction gain control
109    double F_default = 0.0;
110
111    // Init ddynamic_reconfigure to tune gains and set default values
112    ddr.reset(new ddynamic_reconfigure::DDynamicReconfigure(controller_nh));
113
```

```
114   // If it is a Cartesian controller, change matrix gain to a cartesian one
115   if(controller_type.compare("Cartesian_CT") == 0 or controller_type.compare("Optimal") == 0)
116   {
117     // Resize gain matrixs
118     Kp.resize(6,6);
119     Kp = Eigen::MatrixXd::Zero(6,6);
120     Kv.resize(6,6);
121     Kv = Eigen::MatrixXd::Zero(6,6);
122
123     for(int i = 0; i < 6; i++)
124     {
125       Kp_vec[i] = kp_default;
126       Kv_vec[i] = kv_default;
127       ddr->RegisterVariable(&Kp_vec[i], controller_nh_ptr_->getNamespace() + "/Kp/" + std::↩
              ↪ to_string(i), 0, 50000);
128       ddr->RegisterVariable(&Kv_vec[i], controller_nh_ptr_->getNamespace() + "/Kv/" + std::to_string↩
              ↪ (i), 0, 5000);
129     }
130   }
131   else
132   {
133     for(int i=0; i<joint_handles_ptr_->size();i++)
134     {
135       Kp_vec[i] = kp_default;
136       Kv_vec[i] = kv_default;
137       F_vec[i] = F_default;
138       ddr->RegisterVariable(&Kp_vec[i], controller_nh_ptr_->getNamespace() + "/Kp/" + ↩
              ↪ joint_names_[i], 0, 50000);
139       ddr->RegisterVariable(&Kv_vec[i], controller_nh_ptr_->getNamespace() + "/Kv/" + ↩
              ↪ joint_names_[i], 0, 5000);
140       ddr->RegisterVariable(&F_vec[i], controller_nh_ptr_->getNamespace() + "/Friction/" + ↩
              ↪ joint_names_[i], -10, 10);
141     }
142
143   }
144
145   // Initializa dynamic reconfigure
146   ddr->publishServicesTopics();
147
148   // Obtain motor parameters from parameter server
149   // Read the actuator parameters from param server
150   for(int i = 0; i < joint_handles_ptr_->size(); i++)
151   {
```

```
152     if (!controller_nh_ptr_->getParam("/motor_params/" + joint_names_[i] + "/↩
            ↪ motor_torque_constant", motor_torque_constant[i]))
153     {
154       ROS_ERROR_STREAM("Could not find motor torque constant for joint " << joint_names_[i]);
155       return false;
156     }
157     if (!controller_nh_ptr_->getParam("/motor_params/" + joint_names_[i] + "/reduction_ratio", ↩
            ↪ reduction_ratio[i]))
158     {
159       ROS_ERROR_STREAM("Could not find reduction ratio for joint " << joint_names_[i]);
160       return false;
161     }
162   }
163
164   // Obtain KDL Tree
165   urdf::Model tiago_model;
166   std::string paramName = "/robot_description";
167   if (!tiago_model.initParam(paramName)){
168     ROS_ERROR("Failed to parse urdf robot model");
169     return false;
170   }
171
172   if (!kdl_parser::treeFromUrdfModel(tiago_model, tiago_kdl)){
173     ROS_ERROR("Failed to construct kdl tree");
174     return false;
175   }
176
177   // Obtain kinematic chain corresponding to TIAGo arm
178   tiago_kdl.getChain("torso_lift_link","arm_7_link",chain);
179
180   // Create publishers
181   error_pub = controller_nh_ptr_->advertise<std_msgs::Float64MultiArray>("position_error",1000);
182   torque_pub = controller_nh_ptr_->advertise<std_msgs::Float64MultiArray>("torques",1000);
183
184   return true;
185 }
186
187 void starting(const ros::Time& /*time*/)
188 {
189   if (!joint_handles_ptr_) {return;}
190
191   //Zero effort commands
192   for (unsigned int i = 0; i < joint_handles_ptr_->size(); ++i)
```

```
193   {
194     (*joint_handles_ptr_)[i].setCommand(0.0);
195   }
196 }
197
198 void stopping(const ros::Time& /*time*/) {
199   //Zero effort commands
200   for (unsigned int i = 0; i < joint_handles_ptr_->size(); ++i)
201   {
202     (*joint_handles_ptr_)[i].setCommand(0.0);
203   }
204 }
205
206 void updateCommand(const ros::Time& /*time*/,
207                    const ros::Duration& period,
208                    const State& desired_state,
209                    const State& state_error)
210 {
211   const unsigned int n_joints = joint_handles_ptr_->size();
212
213   // Preconditions
214   if (!joint_handles_ptr_) {return;}
215   assert(n_joints == state_error.position.size());
216   assert(n_joints == state_error.velocity.size());
217
218   //Obtain current position and velocities, desired state and state errors and store it at KDL::JntArray ↩
        ↪ variables
219   for(unsigned int i=0;i<n_joints;i++)
220   {
221     current.q(i) = (*joint_handles_ptr_)[i].getPosition();
222     current.qdot(i) = (*joint_handles_ptr_)[i].getVelocity();
223     current_torque[i] = (*joint_handles_ptr_)[i].getEffort();
224
225     desired.q(i) = desired_state.position[i];
226     desired.qdot(i) = desired_state.velocity[i];
227     desired.qdotdot(i) = desired_state.acceleration[i];
228
229     error.q(i) = state_error.position[i];
230     error.qdot(i) = state_error.velocity[i];
231
232     // Set gain matrixs
233     Kp(i,i) = Kp_vec[i];
234     Kv(i,i) = Kv_vec[i];
```

```
235       F(i,i) = F_vec[i];
236     }
237
238     // KDL object for calculate dynamic parameters
239     KDL::ChainDynParam dynamics = KDL::ChainDynParam(chain, KDL::Vector(0.0,0.0,-9.81));
240
241     /* ---------- CONTROLLER SELECTION ---------- */
242
243     if(controller_type.compare("Feedforward")==0)
244     {
245       dynamics.JntToGravity(desired.q,G);
246       dynamics.JntToCoriolis(desired.q,desired.qdot,C);
247       dynamics.JntToMass(desired.q,M);
248
249       // Obtain torque needed for each joint
250       tau.data = M.data*desired.qdotdot.data + C.data + G.data - F * current.qdot.data;
251       for (unsigned int i = 0; i < n_joints; ++i)
252       {
253         // Effort command sending
254         const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
255         if(!std::isnan(command))
256           (*joint_handles_ptr_)[i].setCommand(command);
257       }
258
259     }
260     else if(controller_type.compare("PD_Feedforward")==0)
261     {
262       dynamics.JntToGravity(desired.q,G);
263       dynamics.JntToCoriolis(desired.q,desired.qdot,C);
264       dynamics.JntToMass(desired.q,M);
265
266       // Obtain torque needed for each joint
267       tau.data = Kp*error.q.data + Kv*error.qdot.data + M.data*desired.qdotdot.data + C.data + G.data - ↩
            ↪ F * current.qdot.data;
268       for (unsigned int i = 0; i < n_joints; ++i)
269       {
270         // Effort command sending
271         const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
272         if(!std::isnan(command))
273           (*joint_handles_ptr_)[i].setCommand(command);
274       }
275
276     }
```

```
277   else if(controller_type.compare("PD+")==0)
278   {
279     dynamics.JntToGravity(current.q,G);
280     dynamics.JntToCoriolis(current.q,current.qdot,C);
281     dynamics.JntToMass(current.q,M);
282
283     // Obtain torque needed for each joint
284     tau.data = Kp * error.q.data + Kv * error.qdot.data + M.data*desired.qdotdot.data + C.data + G.data ↩
          ↪ - F * current.qdot.data;
285     for (unsigned int i = 0; i < n_joints; ++i)
286     {
287       // Effort command sending
288       const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
289       if(!std::isnan(command))
290         (*joint_handles_ptr_)[i].setCommand(command);
291     }
292
293   }
294   else if(controller_type.compare("Computed_torque")==0)
295   {
296     dynamics.JntToGravity(current.q,G);
297     dynamics.JntToCoriolis(current.q,current.qdot,C);
298     dynamics.JntToMass(current.q,M);
299
300     tau.data = M.data*(desired.qdotdot.data + Kp*error.q.data + Kv*error.qdot.data) + C.data + G.data ↩
          ↪ - F * current.qdot.data;
301     for (unsigned int i = 0; i < n_joints; ++i)
302     {
303       // Effort command sending
304       const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
305       if(!std::isnan(command))
306         (*joint_handles_ptr_)[i].setCommand(command);
307     }
308
309   }
310   else if(controller_type.compare("Cartesian_CT")==0)
311   {
312     dynamics.JntToGravity(current.q,G);
313     dynamics.JntToCoriolis(current.q,current.qdot,C);
314     dynamics.JntToMass(current.q,M);
315
316     // Add extra gains
317     for(int i = n_joints; i < 6; i++)
```

```
318     {
319       Kp(i,i) = Kp_vec[i];
320       Kv(i,i) = Kv_vec[i];
321     }
322
323     // KDL object for computing forward kinematics and obtain end-effector position, velocity and ↩
            ↪ acceleration
324     KDL::ChainFkSolverPos_recursive fk_solver = KDL::ChainFkSolverPos_recursive(chain);
325
326     // Compute forward kinematics for current and desired states
327     fk_solver.JntToCart(current.q, current_cart);
328     fk_solver.JntToCart(desired.q, desired_cart);
329
330     // Compute jacobians
331     // KDL object for obtaining jacobian matrix
332     KDL::ChainJntToJacSolver jacob_solver = KDL::ChainJntToJacSolver(chain);
333     J_prev = J;
334     jacob_solver.JntToJac(current.q,J);
335
336     // Obtain J derivative and pseudoinverse
337     Jdot.data = J.data - J_prev.data; // J difference between time steps
338     J_pinv = J.data.transpose();// * (J.data * J.data.transpose()).inverse(); // Compute pseudo-inverse ↩
            ↪ as J+ = Jt*(J*Jt)^-1
339     // Compute velocity
340     current_vel = J.data * current.qdot.data;
341     desired_vel = J.data * desired.qdot.data;
342     desired_acc = Jdot.data * desired.qdot.data + J.data * desired.qdotdot.data;
343
344
345     /* --------- COMPUTE CARTESIAN ERROR ----------*/
346
347     // Position error
348     error_cart.p = desired_cart.p - current_cart.p;
349
350     // Orientation error
351     double w, x, y, z, wd, xd, yd, zd = 0.0;
352     // Desired orientation
353     desired_cart.M.GetQuaternion(xd, yd, zd, wd);
354     Eigen::Quaterniond desired_orientation = Eigen::Quaterniond(wd, xd, yd, zd);
355     // Current orientation
356     current_cart.M.GetQuaternion(x, y, z, w);
357     Eigen::Quaterniond current_orientation = Eigen::Quaterniond(w, x, y, z);
358     // Orientation error
```

```
359     Eigen::Quaterniond error_orientation = desired_orientation * (current_orientation.inverse());
360     error_orientation.normalize();
361
362     // Store position error in an Eigen::Vector
363     pos_error(0) = error_cart.p.x();
364     pos_error(1) = error_cart.p.y();
365     pos_error(2) = error_cart.p.z();
366     pos_error(3) = error_orientation.x() * error_orientation.w();
367     pos_error(4) = error_orientation.y() * error_orientation.w();
368     pos_error(5) = error_orientation.z() * error_orientation.w();
369
370     // Velocity error
371     vel_error = desired_vel - current_vel;
372
373     // EFFORT COMMAND CALCULUS
374     tau.data = M.data * J_pinv * (desired_acc + Kp * pos_error + Kv * vel_error - Jdot.data * current.↩
            ↪ qdot.data) + C.data + G.data - F * current.qdot.data;
375
376     // Send each effort command to robot joints
377     for (unsigned int i = 0; i < n_joints; ++i)
378     {
379       // Effort command sending
380       const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
381       if(!std::isnan(command))
382         (*joint_handles_ptr_)[i].setCommand(command);
383     }
384
385     // PUBLISH ERROR AND TORQUE FOR CARTESIAN CONTROLLER
386     std_msgs::Float64MultiArray error_msg, torque_msg;
387     error_msg.data.resize(3);
388     torque_msg.data.resize(4);
389
390     for(int i=0;i<4;i++)
391     {
392       if(i<3)
393         error_msg.data[i] = error_cart.p.data[i];
394
395       torque_msg.data[i] = current_torque[i] * (motor_torque_constant[i] * reduction_ratio[i]);
396     }
397
398     error_pub.publish(error_msg);
399     torque_pub.publish(torque_msg);
400
```

```
401     ros::spinOnce();
402
403     return;
404
405   }
406   else if(controller_type.compare("Optimal")==0)
407   {
408     dynamics.JntToGravity(current.q,G);
409     dynamics.JntToCoriolis(current.q,current.qdot,C);
410     dynamics.JntToMass(current.q,M);
411
412     // Add extra gains
413     for(int i = n_joints; i < 6; i++)
414     {
415       Kp(i,i) = Kp_vec[i];
416       Kv(i,i) = Kv_vec[i];
417     }
418
419     // KDL object for computing forward kinematics and obtain end-effector position, velocity and ↩
          ↪ acceleration
420     KDL::ChainFkSolverPos_recursive fk_solver = KDL::ChainFkSolverPos_recursive(chain);
421
422     // Compute forward kinematics for current and desired states
423     fk_solver.JntToCart(current.q, current_cart);
424     fk_solver.JntToCart(desired.q, desired_cart);
425
426     // Compute jacobians
427     // KDL object for obtaining jacobian matrix
428     KDL::ChainJntToJacSolver jacob_solver = KDL::ChainJntToJacSolver(chain);
429     J_prev = J;
430     jacob_solver.JntToJac(current.q,J);
431
432     // Obtain J derivative and pseudoinverse
433     Jdot.data = J.data - J_prev.data; // J difference between time steps
434     J_pinv = J.data.transpose();// * (J.data * J.data.transpose()).inverse(); // Compute pseudo-inverse ↩
          ↪ as J+ = Jt*(J*Jt)^-1
435     // Compute velocity
436     current_vel = J.data * current.qdot.data;
437     desired_vel = J.data * desired.qdot.data;
438     desired_acc = Jdot.data * desired.qdot.data + J.data * desired.qdotdot.data;
439
440
441     /* -------- COMPUTE CARTESIAN ERROR ----------*/
```

```
442
443     // Position error
444     error_cart.p = desired_cart.p - current_cart.p;
445
446     // Orientation error
447     double w, x, y, z, wd, xd, yd, zd = 0.0;
448     // Desired orientation
449     desired_cart.M.GetQuaternion(xd, yd, zd, wd);
450     Eigen::Quaterniond desired_orientation = Eigen::Quaterniond(wd, xd, yd, zd);
451     // Current orientation
452     current_cart.M.GetQuaternion(x, y, z, w);
453     Eigen::Quaterniond current_orientation = Eigen::Quaterniond(w, x, y, z);
454     // Orientation error
455     Eigen::Quaterniond error_orientation = desired_orientation * (current_orientation.inverse());
456     error_orientation.normalize();
457
458     // Store position error in an Eigen::Vector
459     pos_error(0) = error_cart.p.x();
460     pos_error(1) = error_cart.p.y();
461     pos_error(2) = error_cart.p.z();
462     pos_error(3) = error_orientation.x() * error_orientation.w();
463     pos_error(4) = error_orientation.y() * error_orientation.w();
464     pos_error(5) = error_orientation.z() * error_orientation.w();
465
466     // Velocity error
467     vel_error = desired_vel - current_vel;
468
469     // Optimal controller terms
470     Eigen::VectorXd D = - C.data - G.data;
471     Eigen::MatrixXd A = J.data;
472     Eigen::VectorXd b = desired_acc + Kv*vel_error + Kp*pos_error - Jdot.data * current.qdot.data;
473     Eigen::MatrixXd M_inv = M.data.inverse();
474
475     // WEIGHT MATRIX: This is the key term in optimal control. The optimal controller effect will ↩
        ↪ depend on this value
476     //Eigen::MatrixXd W = Eigen::MatrixXd::Identity(7,7);
477     Eigen::MatrixXd W = M_inv;
478     //Eigen::MatrixXd W = (M.data * M.data).inverse();
479     /*Eigen::MatrixXd W = Eigen::MatrixXd::Zero(7,7);
480     W.diagonal() << 1.0, 3.0, 0.5, 1.0, 1.0, 1.0, 1.0;*/
481
482     Eigen::MatrixXd W_inv_sqrt = W.sqrt().inverse();
483
```

```
484     // Control law
485     tau.data = W_inv_sqrt * (A * M_inv * W_inv_sqrt).transpose() * (b - (A * M_inv * D)) - F * ↩
            ↪ current.qdot.data;
486
487     for (unsigned int i = 0; i < 4; ++i)
488     {
489       // Effort command sending
490       const double command = tau(i)/(motor_torque_constant[i] * reduction_ratio[i]);
491       if(!std::isnan(command))
492         (*joint_handles_ptr_)[i].setCommand(command);
493     }
494
495     // PUBLISH ERROR AND TORQUE FOR CARTESIAN CONTROLLER
496     std_msgs::Float64MultiArray error_msg, torque_msg;
497     error_msg.data.resize(3);
498     torque_msg.data.resize(4);
499
500     for(int i=0;i<4;i++)
501     {
502       if(i<3)
503         error_msg.data[i] = error_cart.p.data[i];
504
505       torque_msg.data[i] = current_torque[i] * (motor_torque_constant[i] * reduction_ratio[i]);
506     }
507
508     error_pub.publish(error_msg);
509     torque_pub.publish(torque_msg);
510
511     ros::spinOnce();
512
513     return;
514   }
515   else
516     ROS_ERROR("Invalid controller type for effort_controllers/DirectDynamicController");
517
518   // PUBLISH ERROR AND TORQUE FOR JOINT CONTROLLERS
519   std_msgs::Float64MultiArray error_msg, torque_msg;
520   error_msg.data.resize(4);
521   torque_msg.data.resize(4);
522
523   for(int i=0;i<4;i++)
524   {
525     error_msg.data[i] = error.q(i);
```

```
526       torque_msg.data[i] = current_torque[i] * (motor_torque_constant[i] * reduction_ratio[i]);
527     }
528
529     error_pub.publish(error_msg);
530     torque_pub.publish(torque_msg);
531
532     ros::spinOnce();
533   }
534
535 private:
536   // ROS Control variables
537   std::vector<hardware_interface::JointHandle>* joint_handles_ptr_;
538   ros::NodeHandle* controller_nh_ptr_;
539   std::vector<std::string> joint_names_;
540   std::string controller_type="PD+";
541   ros::Publisher error_pub;
542   ros::Publisher torque_pub;
543
544   static const int JOINTS = 7;
545
546   //Gains - to be updated via rosparam and ddynamic_reconfigure
547   ddynamic_reconfigure::DDynamicReconfigurePtr ddr;
548   double Kp_vec[JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
549   double Kv_vec[JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
550   Eigen::MatrixXd Kp = Eigen::MatrixXd::Zero(JOINTS,JOINTS);
551   Eigen::MatrixXd Kv = Eigen::MatrixXd::Zero(JOINTS,JOINTS);
552
553   // Friction compensation
554   double F_vec[JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
555   Eigen::MatrixXd F = Eigen::MatrixXd::Zero(JOINTS, JOINTS);
556
557   // Motor parameters
558   double motor_torque_constant[JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
559   double reduction_ratio[JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
560
561   // KDL parameters obtained from robot URDF file
562   KDL::Tree tiago_kdl;
563   KDL::Chain chain;
564
565   // Current state
566   KDL::JntArrayAcc current = KDL::JntArrayAcc(JOINTS);
567   KDL::Frame current_cart = KDL::Frame();
568   Eigen::VectorXd current_vel = Eigen::VectorXd::Zero(6);
```

```
569  double current_torque [JOINTS] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
570
571  // Error
572  KDL::JntArrayAcc error = KDL::JntArrayAcc(JOINTS);
573  KDL::Frame error_cart = KDL::Frame();
574  Eigen::VectorXd pos_error = Eigen::VectorXd::Zero(6);
575  Eigen::VectorXd vel_error = Eigen::VectorXd::Zero(6);
576
577  // Desired state
578  KDL::JntArrayAcc desired = KDL::JntArrayAcc(JOINTS);
579  KDL::Frame desired_cart = KDL::Frame();
580  Eigen::VectorXd desired_vel = Eigen::VectorXd::Zero(6);
581  Eigen::VectorXd desired_acc = Eigen::VectorXd::Zero(6);
582
583  // Jacobian
584  KDL::Jacobian J = KDL::Jacobian(JOINTS);
585  KDL::Jacobian J_prev = KDL::Jacobian(JOINTS);
586  KDL::Jacobian Jdot = KDL::Jacobian(JOINTS);
587  Eigen::MatrixXd J_pinv = Eigen::MatrixXd::Zero(JOINTS,6);
588
589  // KDL-obtained dynamic params
590  KDL::JntSpaceInertiaMatrix M = KDL::JntSpaceInertiaMatrix(JOINTS);
591  KDL::JntArray C = KDL::JntArray(JOINTS);
592  KDL::JntArray G = KDL::JntArray(JOINTS);
593
594  // Calculated torque
595  KDL::JntArray tau = KDL::JntArray(JOINTS);
596 };
597
598 #endif
```

B. Anexo II: Código del nodo ROS para requerir una trayectoria

Código B.1: Código C++ del nodo ROS que requiere una trayectoria al robot

```
1
2 // C++ standard headers
3 #include <exception>
4 #include <string>
5 #include <fstream>
6
7 // Boost headers
8 #include <boost/shared_ptr.hpp>
9
10 // ROS headers
11 #include <ros/ros.h>
12 #include <actionlib/client/simple_action_client.h>
13 #include <control_msgs/FollowJointTrajectoryAction.h>
14 #include <ros/topic.h>
15 #include "std_msgs/Float64MultiArray.h"
16
17 // Our Action interface type for moving TIAGo's arm, provided as a typedef for convenience
18 typedef actionlib::SimpleActionClient<control_msgs::FollowJointTrajectoryAction> arm_control_client;
19 typedef boost::shared_ptr< arm_control_client> arm_control_client_Ptr;
20
21
22 // Create a ROS action client to move TIAGo's arm
23 void createArmClient(arm_control_client_Ptr& actionClient)
24 {
25   ROS_INFO("Creating action client to arm controller ...");
26
27   actionClient.reset( new arm_control_client("/arm_dynamic_controller/follow_joint_trajectory") );
28
29   int iterations = 0, max_iterations = 3;
```

```
30  // Wait for arm controller action server to come up
31  while( !actionClient->waitForServer(ros::Duration(2.0)) && ros::ok() && iterations < max_iterations )
32  {
33    ROS_DEBUG("Waiting for the arm_controller_action server to come up");
34    ++iterations;
35  }
36
37  if ( iterations == max_iterations )
38    throw std::runtime_error("Error in createArmClient: arm controller action server not available");
39}
40
41
42// Generates a simple trajectory with two waypoints to move TIAGo's arm
43void waypoints_arm_goal(control_msgs::FollowJointTrajectoryGoal& goal)
44{
45  // The joint names, which apply to all waypoints
46  goal.trajectory.joint_names.push_back("arm_1_joint");
47  goal.trajectory.joint_names.push_back("arm_2_joint");
48  goal.trajectory.joint_names.push_back("arm_3_joint");
49  goal.trajectory.joint_names.push_back("arm_4_joint");
50
51  // Two waypoints in this goal trajectory
52  goal.trajectory.points.resize(2);
53
54  // First trajectory point
55  // Positions
56  int index = 0;
57  goal.trajectory.points[index].positions.resize(4);
58  goal.trajectory.points[index].positions[0] = 0.2;
59  goal.trajectory.points[index].positions[1] = 0.0;
60  goal.trajectory.points[index].positions[2] = -1.5;
61  goal.trajectory.points[index].positions[3] = 1.94;
62
63  // Velocities
64  goal.trajectory.points[index].velocities.resize(4);
65  for (int j = 0; j < 4; ++j)
66  {
67    goal.trajectory.points[index].velocities[j] = 0.5;
68  }
69  // To be reached 2 second after starting along the trajectory
70  goal.trajectory.points[index].time_from_start = ros::Duration(2.0);
71
72  // Second trajectory point
```

```
73   // Positions
74   index += 1;
75   goal.trajectory.points[index].positions.resize(4);
76   goal.trajectory.points[index].positions[0] = 2.5;
77   goal.trajectory.points[index].positions[1] = 0.2;
78   goal.trajectory.points[index].positions[2] = -2.1;
79   goal.trajectory.points[index].positions[3] = 1.9;
80   /*goal.trajectory.points[index].positions[4] = 1.0;
81   goal.trajectory.points[index].positions[5] = -0.5;
82   goal.trajectory.points[index].positions[6] = 0.0;*/
83   // Velocities
84   goal.trajectory.points[index].velocities.resize(4);
85   for (int j = 0; j < 4; ++j)
86   {
87     goal.trajectory.points[index].velocities[j] = 1.0;
88   }
89   // To be reached 4 seconds after starting along the trajectory
90   goal.trajectory.points[index].time_from_start = ros::Duration(2.0);
91 }
92
93 std::ofstream error_file;
94 std::vector<double> error_msg;
95 int error_cont = 0;
96 void errorCallback(const std_msgs::Float64MultiArray::ConstPtr& msg)
97 {
98   error_cont++;
99
100  error_file << error_cont << " ";
101  for(int i=0;i<msg->data.size();i++)
102    error_file << msg->data[i] << " ";
103
104  error_file << "\n";
105
106 }
107
108 std::ofstream torque_file;
109 std::vector<double> torque_msg;
110 int torque_cont = 0;
111 void torqueCallback(const std_msgs::Float64MultiArray::ConstPtr& msg)
112 {
113   torque_cont++;
114
115   torque_file << torque_cont << " ";
```

```
116  for(int i=0;i<msg->data.size();i++)
117    torque_file << msg->data[i] << " ";
118
119  torque_file << "\n";
120 }
121
122 // Entry point
123 int main(int argc, char** argv)
124 {
125  // Init the ROS node
126  ros::init(argc, argv, "run_traj_control");
127
128  ROS_INFO("Starting run_traj_control application ...");
129
130  // Precondition: Valid clock
131  ros::NodeHandle nh;
132  if (!ros::Time::waitForValid(ros::WallDuration(10.0))) // NOTE: Important when using simulated clock
133  {
134    ROS_FATAL("Timed-out waiting for valid time.");
135    return EXIT_FAILURE;
136  }
137
138  // Create an arm controller action client to move the TIAGo's arm
139  arm_control_client_Ptr ArmClient;
140  createArmClient(ArmClient);
141
142  // Generates the goal for the TIAGo's arm
143  control_msgs::FollowJointTrajectoryGoal arm_goal;
144  waypoints_arm_goal(arm_goal);
145
146  // Sends the command to start the given trajectory 1s from now
147  arm_goal.trajectory.header.stamp = ros::Time::now() + ros::Duration(1.0);
148  ArmClient->sendGoal(arm_goal);
149
150  // Open data files
151  error_file.open("/tmp/error.data");
152  torque_file.open("/tmp/torques.data");
153
154  ros::Subscriber error_sub = nh.subscribe("/arm_dynamic_controller/position_error", 1000, errorCallback);
155  ros::Subscriber torque_sub = nh.subscribe("/arm_dynamic_controller/torques", 1000, torqueCallback);
156
157  // Wait for trajectory execution
158  while(!(ArmClient->getState().isDone()) && ros::ok())
```

```
159  {
160    ros::spinOnce();
161  }
162
163  error_file.close();
164  torque_file.close();
165
166  return EXIT_SUCCESS;
167}
```